(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 189 173 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
20.03.2002 Bulletin 2002/12

(51) Int Cl.[7]: **G06T 15/50**

(21) Application number: **01307190.7**

(22) Date of filing: **23.08.2001**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE TR
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **23.08.2000 US 227007**
28.11.2000 US 726216

(71) Applicant: **Nintendo Co., Limited**
Minami-ku, Kyoto 601-8501 (JP)

(72) Inventors:
- **Drebin, Robert A.**
California 94301 (US)
- **Yasumoto, Yoshitaka**
Osaka 573-1111 (US)
- **Hollis, Martin**
Cambridge CB2 1RQ (GB)
- **Demers, Eric**
Redwood City, California 94065 (US)

(74) Representative: **Allman, Peter John et al**
MARKS & CLERK,
Sussex House,
83-85 Mosley Street
Manchester M2 3LG (GB)

(54) Achromatic lighting in a graphics system and method

(57) A graphics system (50) including a custom graphics and audio processor (114) produces exciting 2D and 3D graphics and surround sound. The system (50) includes a graphics and audio processor (114) including a 3D graphics pipeline (180) and an audio digital signal processor (156). Cartoon lighting and other non-photorealistic effects can be produced by using a lighting calculation to produce a parameter other than color or opacity for use in a later modification of a color or opacity value. In more detail, the example embodiment uses the lighting calculation to generate texture coordinates used in a texture mapping operation (700a). The texture mapping operation (700a) allows lighting computation results to select between brush strokes for cartoon lighting and other effects. The resulting dynamic cartoon lighting animation can be performed on a low cost platform such as a home video game system or personal computer.

EP 1 189 173 A2

Printed by Jouve, 75001 PARIS (FR)

Description

Field of the Invention

[0001] The present invention relates to computer graphics, and more particularly to interactive graphics systems such as home video game platforms. More particularly, this invention relates to a graphics processing improvement that uses a lighting function to produce a parameter for subsequent use in changing a color or opacity. Still more particularly, the invention relates to a 3D graphics system wherein a lighting calculation defines a parameter such as distance or angle that is applied to a further function (e.g., texturing).

Background And Summary Of The Invention

[0002] Many of us have seen films containing remarkably realistic dinosaurs, aliens, animated toys and other fanciful creatures. Such animations are made possible by computer graphics. Using such techniques, a computer graphics artist can specify how each object should look and how it should change in appearance over time, and a computer then models the objects and displays them on a display such as your television or a computer screen. The computer takes care of performing the many tasks required to make sure that each part of the displayed image is colored and shaped just right based on the position and orientation of each object in a scene, the direction in which light seems to strike each object, the surface texture of each object, and other factors.

[0003] Because computer graphics generation is complex, computer-generated three-dimensional graphics just a few years ago were mostly limited to expensive specialized flight simulators, high-end graphics workstations and supercomputers. The public saw some of the images generated by these computer systems in movies and expensive television advertisements, but most of us couldn't actually interact with the computers doing the graphics generation. All this has changed with the availability of relatively inexpensive 3D graphics platforms such as, for example, the Nintendo 64® and various 3D graphics cards now available for personal computers. It is now possible to interact with exciting 3D animations and simulations on relatively inexpensive computer graphics systems in your home or office.

[0004] It has long been known to perform lighting calculations in a 3-D graphics systems based on a variety of parameters (e.g., distance attenuation, angle, beam lighting, etc.). Traditionally, the results of such lighting calculations were used to modify the color and/or opacity of the object being displayed -- as is documented in a variety of standard reference materials on computer graphics (see for example, Foley et al, Computer Graphics Principles and Practice (2d. Ed. 1990) at Chapter 16 ("Illumination and Shading"); and Moller et al, Real Time Rendering (AK Peters 1999) at Section 4.3 et seq. ("Lighting and Shading"); Rogers et al, Procedural Elements for Computer Graphics (2d Ed. McGraw-Hill 1997) at Section 5.2 et. seq. ("Illumination Models"); Neider et al, OpenGL Programming Guide (Addison-Wesley 1993) at Chapter 6 ("Lighting"); and Kovach, Inside Direct3D (Microsoft Press 2000) at Chapter 5 ("Direct3D Vertices and the Transformation and Lighting Pipeline"). For example, techniques known as Gouraud shading and Phong shading modify the color of a displayed surface depending on a lighting effect modeled by a lighting equation. As one example, shining a bright spotlight on a shiny surface can have the effect of whitening the surface color at points where the light is shining. This is often accomplished in a conventional graphics system by having the rasterizer determine the color of each pixel of the displayed surface of a primitive based on the primitive's color (often defined on a vertex-by-vertex basis) and on the result of the lighting equation(s).

[0005] A problem graphics system designers confronted in the past was how to efficiently create non-photorealistic images such as cartoon characters. For many years, most of the work in the graphics field was devoted to creating images that are as realistic as photographs. More recently, however, there has been an interest in non-photorealistic imaging.

[0006] One type of non-photorealistic imaging that has recently generated interest is the process of automating the imaging of cartoon characters. During the heyday of hand drawn cartooning in the 1930s and 1940s, artists created wonderful cartoon characters that dynamically changed from frame to frame. These hand drawn cartoons in many ways set the standard for cartoon rendering. The cartoons were not intended to appear realistic. To the contrary, the cartoons were designed to appear as caricatures. For example, in such cartoons, a person's face might appear to be pasty white with rosy red cheeks defined by brilliantly red or reddish-pink coloration. As the character moved through the scene, the cheek coloration might change dynamically as the artist hand-colored each frame. Such dynamic effects are fascinating to watch and add interest to the cartoon images.

[0007] Unfortunately, the hand-drawn cartoon artistry of the 1930s and 1940s was very time consuming and expensive. Moreover, people now want to use video and computer games to interact with cartoon characters. While video games have for a number of years been successful in dynamically rendering cartoon characters interactively, they have never achieved the hand-drawn artistry details of latter-day hand-drawn cartoons. While much work has been done on high end type systems for authoring cartoon and other non-photorealistic images, further improvements are possible and desirable.

EPODOC ID: <EP_____1189173A2_I_>

[0008] The present invention provides improvements in non-photorealistic and other imaging effects that can be implemented using a low cost graphics system such as, for example, a home video game platform or a personal computer graphics accelerator.

[0009] In accordance with one aspect provided by this invention, the lighting function of the type typically used to light objects within a scene is used to produce a parameter other than color. Such a parameter is used to modify a color or opacity of an object.

[0010] In accordance with another aspect provided by this invention, a lighting calculation performs per-vertex lighting to provide conventional color component outputs. The color component outputs are applied to a texturing function that processes the color components as achromatic parameters. The texturing function output provides a visualization effect (e.g., color and/or opacity modified based on the achromatic parameters) that is used to contribute to the visualization of a rendered scene.

[0011] In accordance with yet another aspect provided by this invention, the lighting calculation output provides three color components one of which is discarded. The other two color components are converted into texture coordinates and used in a texture mapping operation. One of the texture coordinates selects a one-dimensional texture map within a two-dimensional table. The other texture coordinate selects a particular texel within the selected one-dimensional texture map The resulting selected texel provides color and/or alpha information that is applied to a surface defined by the vertex

[0012] In accordance with yet another aspect provided by this invention, non-photorealistic cartoon lighting effects are provided in a real time 3D graphics system by using a lighting calculation to produce a parameter other than color or opacity. A one-dimensional texture map can be defined specifying different brush strokes as a function of the parameter. A conventional texture mapping operation can be used to map the one-dimensional texture map onto a polygon surface based on the parameter

[0013] In more detail the disclosed system can provide conventional lighting-based shading (as discussed above) as part of its overall operation but provides an additional enhancement that allows the lighting equation(s) to generate a texture coordinate (i.e. a parameter other than color or opacity). That other parameter may be inputted to a further function (e.g., a lookup table stored as a texture) to provide a further result -- which may in turn change the color/opacity of an object This intermediate step of, having the lighting calculation produce a parameter that only indirectly affects color and/or opacity of a displayed object provides additional flexibility beyond conventional approaches that use the lighting equation output to directly affect primitive surface color.

[0014] A particular example embodiment uses the lighting equation result to select a one-dimensional texture from a number of textures As an example, to display a tree trunk you might define two different one-dimensional textures - one having a range of silvers and the other having a range of browns. The lighting equation could be used to determine the angle at which a light source is striking the tree trunk, with the result selecting between the two textures. The tree trunk could be colored (using conventional texture mapping) with a range of different silvers from an oblique angle but with a range of browns/blacks when the light is striking it head-on. In such a case, the lighting equation selects a one-dimensional texture (i.e. it is acting as a parameter other than color or opacity -- in this particular example, selecting between two different textures)

[0015] In the particular embodiment disclosed herein, improvements are made to the transformation and lighting ("T & L") pipeline to allow it to generate a texture coordinate(s) based on the lighting equation. The transformation and lighting develops the lighting equation output, but instead of providing it directly to change the pixel color/opacity generated by the rasterizer. it provides the result as a texture coordinate to a texture unit. In the disclosed embodiment, the output of the texture unit is a color or opacity that is blended with the object's primitive color/opacity (e.g., as developed by transform and lighting pipeline and the rasterizer in a conventional fashion based on the same or different lighting equation using a conventional shading algorithm such as Gouraud or Phong shading) to provide a modified color/opacity value for display

Brief Description Of The Drawings

[0016] These and other features and advantages provided by the invention will be better and more completely understood by referring to the following detailed description of presently preferred embodiments in conjunction with the drawings of which:

Figure 1 is an overall view of an example interactive computer graphics system;
Figure 2 is a block diagram of the Figure 1 example computer graphics system;
Figure 3 is a block diagram of the example graphics and audio processor shown in Figure 2;
Figure 4 is a block diagram of the example 3D graphics processor shown in Figure 3;
Figure 5 is an example logical flow diagram of the Figure 4 graphics and audio processor;
Figure 6 shows an example high-level image generating process provided by this invention;

Figure 7 shows a more specific image generating process provided by this invention that uses texturing based on an achromatic lighting parameter;
Figure 8A shows an example texture function;
Figure 8B shows an example one-dimensional texture map;
Figure 9 shows an example texture mapping procedure;
Figure 10A shows an example two-dimensional texture map;
Figure 10B shows a further example two-dimensional texture map;
Figure 11 shows an example lighting pipeline implementation;
Figure 12 shows an example association of lights with color channels;
Figure 13 shows an example transform unit implementation;
Figure 14 shows an example lighting pipeline implementation;
Figure 15 shows an example vector dot/add unit implementation;
Figure 16 shows an example normalizer implementation;
Figure 17 shows an example distance attenuator implementation;
Figure 18 shows an example light scaler implementation;
Figure 19 shows an example integer accumulator implementation;
Figure 20 shows an example state transition diagram; and
Figures 21A and 21B show example alternative compatible implementations.

Detailed Description Of Example Embodiments Of The Invention

[0017] Figure 1 shows an example interactive 3D computer graphics system 50. System 50 can be used to play interactive 3D video games with interesting stereo sound. It can also be used for a variety of other applications.

[0018] In this example, system 50 is capable of processing, interactively in real time, a digital representation or model of a three-dimensional world. System 50 can display some or all of the world from any arbitrary viewpoint. For example, system 50 can interactively change the viewpoint in response to real time inputs from handheld controllers 52a, 52b or other input devices. This allows the game player to see the world through the eyes of someone within or outside of the world. System 50 can be used for applications that do not require real time 3D interactive display (e.g., 2D display generation and/or non-interactive display), but the capability of displaying quality 3D images very quickly can be used to create very realistic and exciting game play or other graphical interactions.

[0019] To play a video game or other application using system 50, the user first connects a main unit 54 to his or her color television set 56 or other display device by connecting a cable 58 between the two. Main unit 54 produces both video signals and audio signals for controlling color television set 56. The video signals are what controls the images displayed on the television screen 59, and the audio signals are played back as sound through television stereo loudspeakers 61L, 61R.

[0020] The user also needs to connect main unit 54 to a power source. This power source may be a conventional AC adapter (not shown) that plugs into a standard home electrical wall socket and converts the house current into a lower DC voltage signal suitable for powering the main unit 54. Batteries could be used in other implementations.

[0021] The user may use hand controllers 52a, 52b to control main unit 54. Controls 60 can be used, for example, to specify the direction (up or down, left or right, closer or further away) that a character displayed on television 56 should move within a 3D world. Controls 60 also provide input for other applications (e.g., menu selection, pointer/cursor control, etc.). Controllers 52 can take a variety of forms. In this example, controllers 52 shown each include controls 60 such as joysticks, push buttons and/or directional switches. Controllers 52 may be connected to main unit 54 by cables or wirelessly via electromagnetic (e.g., radio or infrared) waves.

[0022] To play an application such as a game, the user selects an appropriate storage medium 62 storing the video game or other application he or she wants to play, and inserts that storage medium into a slot 64 in main unit 54. Storage medium 62 may, for example, be a specially encoded and/or encrypted optical and/or magnetic disk. The user may operate a power switch 66 to turn on main unit 54 and cause the main unit to begin running the video game or other application based on the software stored in the storage medium 62. The user may operate controllers 52 to provide inputs to main unit 54. For example, operating a control 60 may cause the game or other application to start. Moving other controls 60 can cause animated characters to move in different directions or change the user's point of view in a 3D world. Depending upon the particular software stored within the storage medium 62, the various controls 60 on the controller 52 can perform different functions at different times.

Example Electronics of Overall System

[0023] Figure 2 shows a block diagram of example components of system 50. The primary components include:

- a main processor (CPU) 110,
- a main memory 112, and
- a graphics and audio processor 114.

[0024] In this example, main processor 110 (e.g., an enhanced IBM Power PC 750) receives inputs from handheld controllers 108 (and/or other input devices) via graphics and audio processor 114. Main processor 110 interactively responds to user inputs, and executes a video game or other program supplied, for example, by external storage media 62 via a mass storage access device 106 such as an optical disk drive. As one example, in the context of video game play, main processor 110 can perform collision detection and animation processing in addition to a variety of interactive and control functions.

[0025] In this example, main processor 110 generates 3D graphics and audio commands and sends them to graphics and audio processor 114. The graphics and audio processor 114 processes these commands to generate interesting visual images on display 59 and interesting stereo sound on stereo loudspeakers 61R, 61L or other suitable sound-generating devices.

[0026] Example system 50 includes a video encoder 120 that receives image signals from graphics and audio processor 114 and converts the image signals into analog and/or digital video signals suitable for display on a standard display device such as a computer monitor or home color television set 56. System 50 also includes an audio codec (compressor/decompressor) 122 that compresses and decompresses digitized audio signals and may also convert between digital and analog audio signaling formats as needed. Audio codec 122 can receive audio inputs via a buffer 124 and provide them to graphics and audio processor 114 for processing (e.g., mixing with other audio signals the processor generates and/or receives via a streaming audio output of mass storage access device 106). Graphics and audio processor 114 in this example can store audio related information in an audio memory 126 that is available for audio tasks. Graphics and audio processor 114 provides the resulting audio output signals to audio codec 122 for decompression and conversion to analog signals (e.g., via buffer amplifiers 128L, 128R) so they can be reproduced by loudspeakers 61L, 61R.

[0027] Graphics and audio processor 114 has the ability to communicate with various additional devices that may be present within system 50. For example, a parallel digital bus 130 may be used to communicate with mass storage access device 106 and/or other components. A serial peripheral bus 132 may communicate with a variety of peripheral or other devices including, for example:

- a programmable read-only memory and/or real time clock 134,
- a modem 136 or other networking interface (which may in turn connect system 50 to a telecommunications network 138 such as the Internet or other digital network from/to which program instructions and/or data can be downloaded or uploaded), and
- flash memory 140.

A further external serial bus 142 may be used to communicate with additional expansion memory 144 (e.g., a memory card) or other devices. Connectors may be used to connect various devices to busses 130, 132, 142.

Example Graphics And Audio Processor

[0028] Figure 3 is a block diagram of an example graphics and audio processor 114. Graphics and audio processor 114 in one example may be a single-chip ASIC (application specific integrated circuit). In this example, graphics and audio processor 114 includes:

- a processor interface 150,
- a memory interface/controller 152,
- a 3D graphics processor 154,
- an audio digital signal processor (DSP) 156,
- an audio memory interface 158,
- an audio interface and mixer 160,
- a peripheral controller 162, and
- a display controller 164.

[0029] 3D graphics processor 154 performs graphics processing tasks. Audio digital signal processor 156 performs audio processing tasks. Display controller 164 accesses image information from main memory 112 and provides it to video encoder 120 for display on display device 56. Audio interface and mixer 160 interfaces with audio codec 122, and can also mix audio from different sources (e.g., streaming audio from mass storage access device 106, the output

of audio DSP 156, and external audio input received via audio codec 122). Processor interface 150 provides a data and control interface between main processor 110 and graphics and audio processor 114.

[0030] Memory interface 152 provides a data and control interface between graphics and audio processor 114 and memory 112. In this example, main processor 110 accesses main memory 112 via processor interface 150 and memory interface 152 that are part of graphics and audio processor 114. Peripheral controller 162 provides a data and control interface between graphics and audio processor 114 and the various peripherals mentioned above. Audio memory interface 158 provides an interface with audio memory 126.

Example Graphics Pipeline

[0031] Figure 4 shows a more detailed view of an example 3D graphics processor 154. 3D graphics processor 154 includes, among other things, a command processor 200 and a 3D graphics pipeline 180. Main processor 110 communicates streams of data (e.g., graphics command streams and display lists) to command processor 200. Main processor 110 has a two-level cache 115 to minimize memory latency, and also has a write-gathering buffer 111 for uncached data streams targeted for the graphics and audio processor 114. The write-gathering buffer 111 collects partial cache lines into full cache lines and sends the data out to the graphics and audio processor 114 one cache line at a time for maximum bus usage

[0032] Command processor 200 receives display commands from main processor 110 and parses them -- obtaining any additional data necessary to process them from shared memory 112. The command processor 200 provides a stream of vertex commands to graphics pipeline 180 for 2D and/or 3D processing and rendering. Graphics pipeline 180 generates images based on these commands. The resulting image information may be transferred to main memory 112 for access by display controller/video interface unit 164 -- which displays the frame buffer output of pipeline 180 on display 56

[0033] Figure 5 is a logical flow diagram of graphics processor 154. Main processor 110 may store graphics command streams 210, display lists 212 and vertex arrays 214 in main memory 112, and pass pointers to command processor 200 via bus interface 150. The main processor 110 stores graphics commands in one or more graphics first-in-first-out (FIFO) buffers 210 it allocates in main memory 110. The command processor 200 fetches:

- command streams from main memory 112 via an on-chip FIFO memory buffer 216 that receives and buffers the graphics commands for synchronization/flow control and load balancing,
- display lists 212 from main memory 112 via an on-chip call FIFO memory buffer 218, and
- vertex attributes from the command stream and/or from vertex arrays 214 in main memory 112 via a vertex cache 220.

[0034] Command processor 200 performs command processing operations 200a that convert attribute types to floating point format, and pass the resulting complete vertex polygon data to graphics pipeline 180 for rendering/rasterization. A programmable memory arbitration circuitry 130 (see Figure 4) arbitrates access to shared main memory 112 between graphics pipeline 180, command processor 200 and display controller/video interface unit 164.

[0035] Figure 4 shows that graphics pipeline 180 may include:

- a transform unit 300.
- a setup/rasterizer 400.
- a texture unit 500.
- a texture environment unit 600, and
- a pixel engine 700.

[0036] Transform unit 300 performs a variety of 2D and 3D transform and other operations 300a (see Figure 5). Transform unit 300 may include one or more matrix memories 300b for storing matrices used in transformation processing 300a. Transform unit 300 transforms incoming geometry per vertex from object space to screen space; and transforms incoming texture coordinates and computes projective texture coordinates (300c). Transform unit 300 may also perform polygon clipping/culling 300d. Lighting processing 300e also performed by transform unit 300b provides per vertex lighting computations for up to eight independent lights in one example embodiment. Transform unit 300 can also perform texture coordinate generation (300c) for embossed type bump mapping effects, as well as polygon clipping/culling operations (300d).

[0037] Setup/rasterizer 400 includes a setup unit which receives vertex data from transform unit 300 and sends triangle setup information to one or more rasterizer units (400b) performing edge rasterization, texture coordinate rasterization and color rasterization

[0038] Texture unit 500 (which may include an on-chip texture memory (TMEM) 502) performs various tasks related

to texturing including for example:

- retrieving textures 504 from main memory 112,
- texture processing (500a) including, for example, multi-texture handling, post-cache texture decompression, texture filtering, embossing, shadows and lighting through the use of projective textures, and BLIT with alpha transparency and depth,
- bump map processing for computing texture coordinate displacements for bump mapping, pseudo texture and texture tiling effects (500b), and
- indirect texture processing (500c).

[0039] Texture unit 500 outputs filtered texture values to the texture environment unit 600 for texture environment processing (600a). Texture environment unit 600 blends polygon and texture color/alpha/depth, and can also perform texture fog processing (600b) to achieve inverse range based fog effects. Texture environment unit 600 can provide multiple stages to perform a variety of other interesting environment-related functions based for example on color/alpha modulation, embossing, detail texturing, texture swapping, clamping, and depth blending.

[0040] Pixel engine 700 performs depth (z) compare (700a) and pixel blending (700b). In this example, pixel engine 700 stores data into an embedded (on-chip) frame buffer memory 702. Graphics pipeline 180 may include one or more embedded DRAM memories 702 to store frame buffer and/or texture information locally. Z compares 700a' can also be performed at an earlier stage in the graphics pipeline 180 depending on the rendering mode currently in effect (e.g. z compares can be performed earlier if alpha blending is not required). The pixel engine 700 includes a copy operation 700c that periodically writes on-chip frame buffer 702 to main memory 112 for access by display/video interface unit 164. This copy operation 700c can also be used to copy embedded frame buffer 702 contents to textures in the main memory 112 for dynamic texture synthesis effects. Anti-aliasing and other filtering can be performed during the copy out operation. The frame buffer output of graphics pipeline 180 (which is ultimately stored in main memory 112) is read each frame by display/video interface unit 164. Display controller/video interface 164 provides digital RGB pixel values for display on display 102.

Example Achromatic Lighting Function

[0041] As discussed above, transform unit 300 in the example embodiment performs lighting in addition to geometric transformations, clipping, culling and other functions. In the example embodiment, transform unit 300 supports lighting in hardware as a per-vertex calculation. This means that a color (RGB) value can be computed for every lit vertex, and that these colors can be linearly interpolated over the surface of each lit triangle. This is known as Gouraud shading. Additional details concerning an example lighting implementation are discussed below in connection with Figures 12 and following.

[0042] The example embodiment transform unit 300 provides an additional use for lighting function(s) 302 not available in traditional graphics systems. Generally, the output of the lighting function in the traditional graphics system is a color used for polygon shading. Some graphics systems provide an alpha (e.g., transparency) value based on a lighting function. In the example embodiment provided by the present invention, a lighting function is used to provide an achromatic parameter (i.e., neither color nor alpha, but something else) used to subsequently modify a color and/or opacity.

[0043] Figure 6 is a block diagram showing such an arrangement where lighting function 302 provides an achromatic parameter used by a subsequent process 304 to modify a color or opacity. The results of the modification are displayed on a display device such as display 56. In this context, the term "lighting function" generally means a mathematical function of a surface position, a surface orientation, an eye location and, in some cases, a light location. Lighting functions can be defined as any combination of these coordinates.

[0044] In the example embodiment, the achromatic parameter outputted by lighting function 302 is processed by a texturing function 306 as shown in Figure 7. At a very simple level, texturing an object means "glueing" an image onto that object. In more detail, texturing generally works by modifying the values used in lighting function 302. For example, in the typical case, a lighting function 302 is used to determine the color and illumination of a polygon surface, and a texturing function is used to modify that color/illumination to provide an additional effect (e.g., to create the appearance of a brick wall or a wood-grain, to make the surface appear bumpy, mirrored or warped, etc.). In the example embodiment shown in Figure 7, in contrast, the output of lighting function 302 is not a color or transparency value, but is instead a generalized parameter (for example, a texture coordinate) for texturing function 306. The texturing function develops color and/or opacity information used to modify (e.g., blend) with a primitive color and/or opacity for display on display device 56.

[0045] Figure 8A shows an example texture function 306. In this example, the lighting function 302 produces an achromatic output parameter S that parameterizes the example texture function 306. In this Figure 8A example, the

example texture function 306 selects one of three colors (C1, C2, C3) or opacities depending upon the value of parameter S. Thus, the particular color or opacity outputted by texture function 306 is determined as a function of the parameter S outputted by lighting function 302. Because lighting function 302 is providing parameter S. the selection of which color or opacity outputted by texture function 306 can depend on any of various factors used by the lighting function calculation, including for example:

- distance,
- attenuation,
- angle,
- vector,
- cosine,
- polynomial,
- light source type,
- other coefficient.

[0046] While texture function 306 may be implemented in a variety of different ways, a cost-effective approach is to store the texture function 306 output values in a texture lookup table or map and to access those values using texture coordinates. Figure 8B shows an example one-dimensional texture map 308 indexed by lighting function 302 output parameter S. An advantage of using a conventional texture mapping function in the example embodiment is that texture mapping hardware or other functionality available for general purpose texturing can also be used to implement texture function 306 in cooperation with lighting function 302.

[0047] Figure 9 shows a more detailed example including a two-dimensional texture mapping operation 306. In this example, a lighting calculation 302 is performed based on an identification of vertices and lighting definitions. The output of lighting calculation 302 in the example embodiment comprises per-vertex colors (e.g., red, green, blue) and/or transparency (i.e., alpha). The example embodiment typically uses these lighting calculation 302 color outputs for Gouraud shading of a polygon surface. However, in this mode of operation, one of the color channels (e.g., blue) is discarded, and the other two color channels (i.e., red, green) are converted into texture coordinates s, t using a conversion operation 308. The texture coordinates s, t are applied to one or more texture mapping operations 306a, 306b, etc. to provide color/transparency output texels. Blending operation 602 may blend these texels with surface color information (which may be derived using conventional shading techniques based on the same or different lighting calculation) to provide shaded, textured polygons for display.

[0048] In the example embodiment, texture mapping operation 306 provides two-dimensional texture mapping as shown in Figure 10A. In this example, the t texture coordinate is generated based on the green color channel output of lighting calculation 302, and is used to select one of n one-dimensional texture maps within a two-dimensional texture map 310. The other texture coordinate s is developed based on the red color channel output of lighting calculation 302, and is used to select a particular texel within the one-dimensional texture selected by the t coordinate. Of course, the s texture coordinate could select the 1-D map and the t texture coordinate could select a particular texel in other implementations, and/or different colors and/or opacities could be used to generate the different texture coordinates. Three texture coordinates could be used to select texels in a 3D texture if desired.

[0049] Figure 10B is an illustrative example of a two-dimensional texture map 310 comprising a number of one-dimensional texture maps 308. The particular two-dimensional texture map shown in Figure 10B is particularly suitable for cartoon lighting effects. The example shown includes a number of one-dimensional texture maps including four maps 308(1) ... 308(4) each comprising two different kinds of texels (e.g., purple texels and blue texels). One of the texture coordinates (e.g., t) selects between these different texture maps and the other texture coordinate (e.g., s) selects a particular texel (i.e., blue or purple). The various 1D texture maps 308(1) ... 308(4) provide different mappings between the texture coordinates and blue or purple texels to provide different brush strokes or other effects.

[0050] As another example, the 1D texture map 308(6) shown in Figure 10B includes four different types of texels (yellow, orange, red and brown). These different colors could be used to provide bold cartoon-like lighting effects where, for example, the angle a directional light makes with an object or the distance of an object from a light source determines the color resulting from the output of texture mapping operation 306. Such visualization can have a variety of interesting applications - especially in non-photorealistic real time rendering such as dynamically-generated cartoon animation. For example, it is possible to define a virtual cartoon light which lights up an object. The object's rendered visualization becomes dependent on vertex position, the local cartoon light position and other factors the lighting calculation 302 takes into account (e.g., specular or diffuse computation, distance attenuation, etc.). In one example, it is possible to set the red color channel to zero and allow the lighting calculation 302 to compute the green channel to specify a particular texel value within a 1D texture defining a set of brush strokes. The material color main processor 110 typically can apply to lighting calculation 302 can be used to specify which 1D texture to select within a 2D texture map. Lighting calculation 302 calculates which brush stroke to select on a vertex-by-vertex basis at no extra cost beyond supplying

one index per vertex.

[0051] While the example shown in Figure 9 is particularly useful for cartoon lighting, it can also be used for many other applications. As one example, lighting function 302 can be used to calculate a shadow volume or surface. Lighting calculation 302 based on a given vertex could generate a shadow volume defined at texture coordinates used for the texture mapping operation 306. Many other applications are possible. For example, one interesting application relates to projective textures. An undesirable property of projective textures is that they typically project forward and backward from the camera. The techniques disclosed herein can provide projective texture coordinate generation based on a lighting function which provides attenuation (e.g., to eliminate backwards facing lighting) and/or distance attenuation (e.g., to attenuate the projective texture based on how far it is projected - just as you might see when using a real slide projector to project an image. Such projective texturing in combination with the achromatic lighting function described herein can produce many interesting images and imaging effects.

[0052] A conversion operation 306 as shown in Figure 9 may not be necessary in all implementations. In the example embodiment, lighting calculation 302 generates an integer output that conversion block 308 converts into a floating point representation suitable for texture coordinates used in the following texture mapping operation 306. However, such integer-to-floating point conversion would not be necessary in other implementations, or other implementations could use different types of conversions as desired.

[0053] In the example embodiment shown in Figure 9, lighting calculation 302 need not calculate both s and t texture coordinates. Since only one texture coordinate is used as a parameter in the example embodiment to select which particular texel was in a one-dimensional texture map should be applied to a primitive surface, the preferred embodiment preferably calculates that texture coordinate (or in intermediate value from which that texture coordinate is derived) using lighting calculation 302. However, depending on the application, it might be desirable for the application program as opposed to lighting calculation 302 to specify another texture coordinate used to select (in this example embodiment) between plural one-dimensional textures 308. As one example, the application program running on main processor 110 can specify a value for any or all of the color channel outputs of lighting calculation 302 via the lighting definitions applied to the lighting calculation input. In some applications, it may be desirable for the main processor 110 to specify the contents of, for example, the green color channel and thus the particular one-dimensional texture used in the texture mapping operation 306. This provides additional application programmer control while still allowing the lighting calculation 302 to dynamically generate the s texture coordinate based on the variables lighting calculation unit 302 evaluates. Floating point values the main processor 110 sends to transform unit 300 can be truncated to RGB8 to allow calculations to occur in higher precision.

[0054] In the example embodiment, there is no reason why lighting calculation 302 cannot produce a negative output value. In the example shown in Figure 9, this corresponds to backlighting an object. Thus, it is possible to have negative or backlit cartoon lighting in the example embodiment. This could be useful to provide effects where there is information on the backside of an object. In the preferred embodiment, the lighting unit always outputs a positive color (value). The preferred embodiment supports a negative type of lighting calculation, but it up to the application to add a scale and/or a bias so that the final value ends up being positive. For example, the output of the lighting calculation could be mapped from -1 to 0 and from +1 to 1 before being converted to a texture coordinate(s). The example embodiment does not clamp negative values, but simply maps them into positive numerical values - with the application being careful to ensure that these positive values are interpreted appropriately. Of course, other implementations could support negative lighting values directly if desired.

[0055] Texture mapping operation 306 and blending operation 602 shown in Figure 9 can be arbitrarily complex. For example, it is possible to blend with alpha. As another example, it will be possible to use two separate color channels outputted by lighting calculation 302 for two different texture mapping operations which could then be blended together by blender 602. Since texture mapper 500 and texture environment unit 600 in the example embodiment are both multi-task/multi-stage operations, a sequence of direct/indirect operations can be provided based on the output of lighting calculation 302 to provide a variety of interesting and complicated effects.

Example Lighting Pipeline Implementation

[0056] Figure 11 shows an example block diagram of lighting calculation 302. In this example, lighting calculation 302 is performed by transform unit 300 in response to information received from command processor 200. This information can come from a variety of sources including main processor 110 and main memory 112 (see Figure 4).

[0057] In the example embodiment, the transform unit 300 includes a master control section 320, a lighting computation pipeline 322, an accumulator 324 and - for purposes of texture generation, an integer-to-floating point converter 308. Master controller 320 receives lighting definitions and vertex definitions from command processor 200 and, after appropriate storage/buffering, provides associated information to the lighting computation pipeline 322. Lighting pipeline 322 performs a lighting computation in the following basic form:

$$C_n = C_{material}[A_c + \sum_{i=0} L(c) \cdot A]$$

where C_n defines the output of the lighting calculation pipeline, $C_{material}$ defines the material color, A_c defines the ambient color, L(c) defines the lighting diffuse or specular component, and A defines distance attenuation. This calculation computes an RGB triplet since the example embodiment lighting calculation pipeline 322 can perform two such lighting calculations in parallel. Because of this parallel computation feature, one of the lighting computations can be used for achromatic lighting effects for an object and the other lighting computation can be used for chromatic lighting effects on the same object. The parallel computation feature makes effects such as cartoon lighting "free" in the sense that it takes no more processing time to generate a shaded polygon surface with cartoon lighting effects than it does to generate the shaded polygon surface without such cartoon lighting effects. In the example embodiment, the C_n value generated by lighting computation pipeline 322 can be positive or negative - with negative lighting allowing backlit and other interesting effects.

[0058] In the example embodiment, a C_n output of lighting computation pipeline 322 is accumulated by a integer accumulator 324. The accumulated output is, in the example embodiment, converted into floating point by conversion block 308 and provided as texture coordinates to texture unit 502 for texture mapping operations. The example embodiment can provide texture filtering as part of texture mapping operation 306. The application programmer should be careful about the texture filtering mode chosen when using lighting calculation 302 to generate texture coordinates. Problems may result due to the fact that the s and t axes in the example embodiment are used for independent factors, but certain types of texture filtering apply to both axes simultaneously. One work around is to duplicate, within the texture map, each entry along the t axis in order to ensure that interpolation between adjacent t values has no effect on the final output. Thus, one-dimensional texture filtering for adjacent values of t will be identical. Mipmap filtering with multiple levels of details can also take place, but this may also act to reduce the number of 1D textures within a given 2D texture map.

[0059] In the example embodiment, another complication results from the method of converting the lighting calculation 302 color value into texture coordinates. The color value produced by lighting calculation 302 in the example implementation is an 8-bit integer in the range of 0-255. Convert block 308 converts this integer value into a floating point number by dividing it by 255. However, the value is converted into a texture coordinate by multiplying it by the texture size in the example embodiment. This process means that the application programmer should pay careful attention to the texture coordinate for choosing the 1D texture. Assuming a texture size of 256 and GX_NEAR texture filtering, this will map in the following manner:

Color Value	Converted texture coordinate	Nearest value
0	0	0
64	64.251	64
127	127.498	127
128	128.502	129
192	192.753	193
254	254.996	255
255	256	255

[0060] Due to the conversion process, coordinate value 128 is skipped, and color values 254 and 255 map to the same coordinate value. If we assume a texture size of 256 and GX_LINEAR filtering, we have the following mappings:

Color Value	Converted texture coordinate	Coords looked up
0	-0.5	0,0
1	0.504	0, 1
2	1.508	1, 2
3	2.512	2, 3
4	3.516	3, 4

(continued)

Color Value	Converted texture coordinate	Coords looked up
126	125.994	125, 126
127	126.998	126, 127
128	128.002	128, 129
253	253,492	253, 254
254	254.496	254, 255
255	255.5	255, 255

[0061] It is safe to use n*2 to convert a table ID into a color value in the example implementation. However, the 1D textures should be stored in a non-straightforward manner within the 2D texture. The table for n=0 should be stored at coordinate 0 (only), tables for n=1 to n=63 should be stored at n*2-1 and n*2, and tables for n=64 to n=127 should be stored at n*2 and n*2+1. Coordinate 127 may be left empty since it will not normally be accessed in this particular implementation. Other implementations might avoid this issue altogether, or present different conversion issues.

Example More Detailed Transform Unit 300 Description

[0062] Figures 12-20 show a further example, more detailed embodiment of transform unit 300 including a lighting pipeline used for lighting calculation 302. In the example embodiment of system 50, transform unit 300 supports lighting in hardware as a per-vertex calculation. This means that a color (RGB) value can be computed for every lit vertex, and that these colors can be linearly interpolated over the surface of each lit triangle (known as Gouraud shading). Transform unit 300 in this example embodiment has full support for diffuse local spotlights, and also has some support for infinite specular lighting.

[0063] Transform unit 300 in this example embodiment supports diffuse attenuation. This means that the front of the object can be brighter than the sides, and the back darkest. Transform unit 300 supports vertex normals so as to provide diffuse attenuation. For each vertex, the vertex normal (N) is compared against the vector between the vertex and the light position.

[0064] The example embodiment of transform unit 300 also supports local lights. Local lights have a position within the world and possibly also a direction. Each light is defined to have a position. Using the position of each vertex and the position the light, transform unit 300 can perform per-vertex distance attenuation. This means you can make the brightness of the light shining on an object decrease as the object moves away from the light.

[0065] Transform unit 300 in the example embodiment also supports directional lighting. Support ranges non-directional lights, to subtle directional effects, to highly directional spotlights. These effects are supported by angle attenuation. Thus, vertices directly "in the beam" of the light can be made brighter than vertices outside of the beam or behind the light.

[0066] Local diffuse lights can be both distance-attenuated and angle-attenuated. By programming the proper lighting equation, it is possible for an application programmer to obtain attenuation values as an output color or alpha (or, in the case of texture coordinate generation, as a texture coordinate).

[0067] In the example implementation, transform unit supports eight physical lights. The application programmer can describe the attenuation parameters, position, direction and color of each light. The application programmer can also control up to four physical color channels that accumulate the result of the lighting equation. By associating lights with channels, the application programmer can choose to sum the effect of multiple lights per vertex, or combine them later in the texture environment unit 600. The number of channels available to the texture environment unit 600 is set by the application programmer. In some cases (e.g., when using a color channel to generate texture coordinates), a light channel is computed but not outputted as a color or opacity. As discussed above, transform unit 300 is pipelined so as calculate two color channels simultaneously, but may provide only one color channel directly to texture environment unit 600 for blending. The other color channel can be provided in the form of texture coordinates to texture unit 500.

[0068] Each color channel enables attenuation in the selection of color source. A light mask associates up to eight lights with the channel. See Figure 12 channel which shows example association of up to eight different lights with any of two color channels and two alpha channels to provide two independent outputs one of which can be converted to texture coordinates.

[0069] As shown in Figure 12, transform unit 300 in the example embodiment generates two RGBA colors (color 0 and color 1). Each output color has two lighting functions: one for RGB and one for alpha, for a total of four lighting functions per polygon per vertex. Four such lights allow a variety of lighting effects such as, for example, in multitex-

turing. Each function can be comprised of a material color in operations based on a global ambient color and the state of the eight lights retained by transform unit 300. The equations enable diffuse, specular and spotlight attenuation. The lighting data path is designed to be optimal for local diffuse spotlights in the preferred example embodiment, but it is also possible to generate specular highlights and/or attenuation factors for texture-based lighting with the same data path. As described above, the light colors can also be used to generate texture coordinates.

Example Lighting Calculation

[0070] Example lighting equations performed by transform unit 300 are as follows:

$$C = \begin{cases} Material_c \begin{bmatrix} R \\ G \\ B \end{bmatrix} x\ LightFunc \begin{bmatrix} R \\ G \\ B \end{bmatrix}, if\ C = Color_{0_{RGB}}, Color_{1_{RGB}} \\ Material_c[A] \times LightFunc_c[A], if\ C = Color_{0_{Alpha}}, Color_{1_{Alpha}} \end{cases}$$

$$Material_c = MaterialSrc_c = REGISTER?\ VertexColor_c : Material\ \mathrm{Reg}_c$$

$$LightFunc_c = \begin{cases} 1.0, & if\ LightFuncEnable_c = FALSE \\ Illum_c, & if\ LightFuncEnable_c = TRUE \end{cases}$$

$$Illum_c = Clamp\left(Amb_c + SignedInt\left(\sum_{i=0}^{i=7} Enable_c(i) Atten_c(i) DiffuseAtten_c(i) Color_i \right) \right)$$

$$Amb_c = (AmbSrc_c = REGISTER\ ?\ VertexColor_c : Ambient\ \mathrm{Reg})$$

[0071] In the example embodiment, the material and global ambient colors can come from a register or in the form of a vertex color from command processor 200. An application requiring more than eight lights can compute the illumination from the additional lights, based on software executing on main processor 110, so that the result is one of the vertex colors, and set the ambient source register to use that vertex color. It is also possible if desired for main processor 110 to compute any or all of the lighting calculations using software and provide those resulting computation outputs in the form of vertex colors to transform unit 300 for conversion into texture coordinates; or the main processor may pass the computed texture coordinates to texture unit 500 for purposes of texture mapping.

[0072] Disabling the "LightFunc" parameter passes the material color through transform unit 300 unchanged. This can be used to allow main processor 110 to directly select, based on application program software control, which of a plurality 1D textures to use in a texture mapping operation responsive to an achromatic lighting function output.

[0073] Any or all of the available eight lights can be enabled in each lighting function. The sum of the per-light illumination is clamped to [-1, 1] in the example embodiment, and converted to integer 2's complement before adding the global ambient term. Since the total illumination is clamped to [0, 1], a material cannot become brighter through lighting in the example embodiment. If this effect is desired, the light colors can be premultiplied by the material color, and the material color in the equation can be set to 1.0. Other arrangements can be provided in other example implementations to allow materials to become brighter through lighting.

[0074] In the example embodiment, the alpha equations can be used when monochrome results are sufficient. The resulting alpha values can be combined with other RGB and alpha values in texture environment unit 600 in the example embodiment. For example, it is possible to put the diffuse result in the color output for the channel 0 color, a specular result in the alpha output of the color 0 channel, and have the texture environment unit 600 multiply the diffuse color by the texture color and add a (e.g., white) specular highlight in a single stage of texture environment unit 600.

[0075] Flexibility is provided in the diffuse attenuation function in the example embodiment, beyond the command (N H) clamp to [0, 1]. Attenuation can be defeated for lighting equations which do not have a diffuse property or left

EPODOC <EP_____1189173A2_I_>

unclamped. Unclamped dot products allow light-based texture generation to control illumination for 180°. Since the illumination will be clamped in the example embodiment, signed lighting functions may be scaled and biased appropriately using $color_c$ and amb_c. The example embodiment provides three example diffuse attenuation functions a lighting equation can use:

$$DiffuseAtten_c(i) = \begin{cases} 1.0, & if\ DiffAttenSelect_c = NONE \\ \hat{N} \cdot \hat{L}_i, & if\ DiffAttenSelect_c = SIGNED\ (if\ SPOTLIGHT\ only) \\ Clamp0(\hat{N} \cdot \hat{L}_i), & if\ DiffAttenSelect_c = CLAMP(if\ SPOTLIGHT\ only) \end{cases}$$

[0076] The angle attenuation logic in the example embodiment computes a second-order polynomial based on the dot product of the light-to-vertex vector and the light-direction vector. A sharp fall off is achieved by extrapolating the squared and linear terms. Clamping is used to avoid negative values produced by extrapolation for angles outside of a spotlight angle. Range attenuation can be performed by the inverse of another second-order polynomial. The distance value, d, is the length of the vector from the vertex to the light position. A light equation with only distance attenuation can be used for texture-based lights to simulate the distance-based illumination falloff of projected or other textures.

[0077] This same logic can be used to approximate common specular attenuation, $(N \cdot H)^S$ for a parallel light source. The light-to-vertex and light-direction vectors may be replaced with the normal and half-angle vectors respectively. Attenuation can be defeated in a lighting function to permit non-attenuated point light (e.g., omni-direction) sources to be used in both a diffuse and specular equation. In this example, the light's angle attenuation coefficients may be used for the specular equation, and attenuation is turned off in the diffuse equation. Specular and diffuse equations in the example embodiment are combined in texture environmer 600 as opposed to within transform unit 300 in the example embodiment, but other implementations are possible.

[0078] Further details of the example attenuation function are as follows:

$$Atten_c = AttenEnable_c ? \frac{Clamp0(a_{2_i} AAtt_c(i)^2 + A_{1_i} AAtt_c(i) + A_{0_i})}{K_{2_i} d_c^2(i) + K_{1_i} d_c(i) + K_{0_i}} : 1.0$$

$$AAtt_c(i) = \begin{cases} \hat{N} \cdot \hat{L} > 0 ? clamp0(\hat{N} \cdot H_i) : 0, if & AttenSelect_c = SPECULAR \\ clamp0(\hat{L} \cdot L_{dir_i}), if & AttenSelect_c = SPOTLIGHT \end{cases}$$

$$d_c(i) = \begin{cases} \hat{N} \cdot \hat{L} > 0 ? clamp0(\hat{N} \cdot H_i) : 0, if & AttenSelect_c = SPECULAR \\ \sqrt{L_i \cdot L_i}, if & AttenSelect_c = SPOTLIGHT \end{cases}$$

Example Transform Unit 300 Implementation Block Diagram

[0079] Figure 13 is a block diagram of an example implementation of transform unit 300 of system 50. Transform unit 300 in this example embodiment includes three main sections:

- top-of-pipeline section 330,
- light section 332,
- bottom-of-pipeline section 334.

[0080] The top-of-pipeline section 330 in the example embodiment includes light section 332, a context matrix store 336, an input FIFO buffer 338, a dot product unit 340, a projection block 342, a texture dot two block 346, a cripping detector 348, and a bypass FIFO block 350. In the example embodiment, top-of-pipeline (TOP) section 330 performs the following example functions:

- vertex transform (3 dot products),
- Normal transform (3 dot products),

SDOCID: <EP_____1189173A2_I_>

- texture transform (2 or 3 dot products for texture),
- projection transform (simplified 3 dot products); and

light section 332 performs the following example color channel functions:

- color channel 0 diffuse computation (1 dot product N·L),
- color channel 0 diffuse computation (1 dot product L^2 per light),
- color channel 0 diffuse computation (1 dot product N^2).

[0081] In addition, the input FIFO receives per-vertex vertex descriptors specifying, for example, the following information:

- geometry information XYZ,
- Normal vector information N_x, N_y, N_z,
- RGBA color 0 per vertex,
- RGBA color 1 per vertex,
- Binormal vector T_x, T_y, T_z,
- Binormal vector B_x' B_y, B_z
- Texture 0 data T_0,
- Texture 1 to n data S_n, T_n.

[0082] Appropriate per-vertex information is provided to light section 332 to enable the lighting computations.

[0083] Light section 332 in the example embodiment shown in Figure 13 includes a lighting parameters memory 352, a Normal memory 354 and a light data path 356.

[0084] Light parameter memory 352 stores various light parameters used by light data path 356. Light parameter store 352 thus holds all of the various lighting information (e.g., light vectors, light parameters, etc.). Both global state and ambient state are stored in this memory in the example embodiment. In the example embodiment, each word is written in 32-bits, but only the twenty most significant bits are kept. Each location is three words wide, with minimum word write size of three words in the example embodiment. The following shows example contents of light parameter memory 352:

Register Address	Definition	Configuration
0/0600	Reserved	
0x0601	Reserved	
0x0602	Reserved	
0x0603	Light0	32b: RGBA (8b/comp)
0x0604	Light0A0	20b: cos atten. A_0
0x0605	Light0A1	20b: cos atten. A_1
0x0606	Light0A2	20b: cos atten. A_2
0x0607	Light0K0	20b: dist atten. K_0
0x0608	Light0K1	20b: dist atten. K_1
0x0609	Light0K2	20b: dist atten. K_2
0x060a	Light0Lpx	20b: x light pos, or inf ldir x
Ox060b	Light0Lpy	20b: y light pos, or inf ldiry y
Ox060c	Light0Lpz	20b: z light pos, or inf ldir z
0x060d	Light0Dx/Hx	20b: light dir x, or 1/2 angle x
0x060e	Light0Dy/Hy	20b: light dir y, or 1/2 angle y
0x060f	Light0Dz/Hz	20b: light dir z, or 1/2 angle z
0x0610-0x067f	Light(n)data	See Light0 data

(continued)

Register Address	Definition	Configuration
0x0680-0x07ff	Not used	Reserved

[0085] An overview of the light section 332 of the example embodiment: is shown in Figure 14. Briefly, the light section 332 performs the following local lighting computations:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = Material_0 \begin{bmatrix} Mr \\ Mg \\ Mb \end{bmatrix}$$

$$(Ambient_0 \begin{bmatrix} Ra \\ Ga \\ Ba \end{bmatrix} + \frac{CosAtFunc_0}{DistAtFunc_0} Clamp(Normal \cdot Light_i) \times Diffuse_i \begin{bmatrix} Rd \\ Gd \\ Bd \\ 0.0 \end{bmatrix})$$

This requires a local light vector computation per vertex:

$$Light_i = Normalized(LightPos_i \begin{bmatrix} Lx \\ Ly \\ Lz \end{bmatrix} - EyeSpaceVertex \begin{bmatrix} Xh \\ Yh \\ Zh \end{bmatrix})$$

after which, we need to compute:

$$DiffuseAtten_0 = Conditional\_clamp\_to\_0(Light_i \cdot EyeSpaceNormal)$$

[0086] Thus, the diffuse attenuation calculation requires the light vector and the eye-space Normal. Clamping to 0 may be conditional on an internal flag (e.g., it can be clamped to 1.0 or not clamped at all) in the example implementation. It is possible to allow diffuse lights to have negative dot products. A lighting dot product unit can compute the eye-space Normal while the transform dot product unit computes the geometry eye-space conversion. Then, a lighting dot product unit can compute the N·L, L^2 and N^2 dot products. After this, the example embodiment normalizes the results to provide normalized light vector Normal information. After normalization, the intermediate results of the normalization (e.g., distance squared and distance and cosine N·L) are used to compute the attenuation equations as well as the normalization factors. Then, the light ambient vector is multiplied. The resulting triplet is the per-light attenuated diffuse component. This is converted to 2's complement integer, and the resulting three values are accumulated in an integer accumulator. This accumulator adds the ambient terms and any other diffuse terms from other lights. The final sum is then the per-vertex color (in the example embodiment, an 8-bit RGB format clamped to 0 to 255).

[0087] Figure 15 shows an example diagram of the vector dot/Madd block 357 shown in Figure 14. If the same dot product unit is used to compute the transform N and all other dot products, then the performances may become somewhat degraded and the scheduling of light becomes difficult since each light needs to re-use the same data path element multiple times. This may lead to complex control issues. To simplify the design, we can implement a second limited dot product unit to separate the various computations. The first dot product unit may compute the transform normal while the other dot product unit computes the vertex transformation. A vector add may then compute the light vector for local lighting. The second dot product may compute N^2, L^2, N·L and N·H (for specular lights). This can be achieved in the example embodiment in a fully pipelined way with no feedbacks. See Figure 15.

[0088] Figure 16 shows an example implementation of the normalizer 360 shown in Figure 14. Once the lighting vector values are computed, the example embodiment normalizes the results and computes attenuation. Example

embodiment normalizer 360 accepts the following inputs:

- Cosine attenuation: L·Dir, 1/sqrt (L)
- Distance attenuation: L^2, L
- Diffuse factor: N·L, N^2, L^2,

and computes distance attenuation using the following step performed by example normalizer 360 shown in Figure 16:

1. Compute K_2d^2
2. Compute $K_2d^2 + K0$
3. Compute d
4. Compute K_1d
5. Compute $D_2D^2 + K_1d + KO$
6. Compute $1/(K_2d^2 + K_1d + K0)$

[0089] Figure 17 shows an example distance attenuation unit 362. The example distance attenuator implementation 362 computes cosine attenuation using the following example steps:

1. Compute Cos = Clamp0(L.Ldir)
2. Compute Cos_2
3. Compute A_2Cos^2
4. Compute A_1Cos
5. Compute $A_1Cos + A0$
6. Compute $A_2Cos^2 + A_1Cos + A0$
7. Compute $Clamp\_0(A+2+Cos^2 + A_1Cos + A0)$

[0090] Figure 18 shows an example implementation of light scaler 364 that computes the following:

- 1/sqrt (N^2)
- 1/sqrt (L^2)
- N·L (sqrt (N^2) x sqrt(L^2)).

[0091] Some of the units (e.g., 1/sqrt and some of the multipliers) can be shared among the various data path implementations shown in Figures 15-18. Thus, an implementation might approximately require on the order of eighteen multipliers and ten adders. Inversion and 1/sqrt can be performed using a table lookup or a simple one-pass Newton-Raphson interpolator or some other pipeline interpolator. Alpha and specular writing computations do not require any additional hardware in the example embodiment since they are just changes in the lighting equation.

[0092] Figure 19 shows an example integer accumulator 336 implementation. When a new attribute arrives for each color, that color is accumulated. Once all lights are accumulated for a color, and the ambient is added (e.g., from the vertex color FIFO or from a register) then the material color is multiplied. The final RGB/A color is then accumulated into an accumulation/format register by construction of the final color and texture generation. Once the color/texture is computed, it is then written into an output FIFO. Control may follow these general guidelines:

```
If (new attribute == new light for color X)
                    Accumulate new light into color X
                    Increment number of lights for color X
End
If (number of lights for color X is maximum)
                    Transfer color to multiplier
                    Perform material multiply
End
If (color X is in multiply state)
                    Accumulate color X first
End
If (Accumulation of color X is complete (RGBX or AXX or RGBA))
                    Format color X for color, if color enabled
                    Format color X for texture, if texture enabled and color disabled or
                    color already loaded
                    Load output fifo with formatted data, increment color/texture count
End
If (color count and texture count at maximum)
                    Write data to BOP
                    Pop fifo
End
```

[0093] Figure 20 shows an example stage transition diagram used to control light pipeline 332 in the example embodiment. Arbiter logic decides the sequence of events that it issued into the light data path. The example embodiment arbiter can only issue a new event every four or eight cycles, depending on the event. The list of possible events includes:

- light computation for a specific color/alpha,
- bump map computation

[0094] For the above events the arbiter drops an attribute into the control pipeline every 4/8 cycles. Each of these attribute carries the execution instructions for the event into the light pipeline 332. At the end of the pipeline, the attributes are used to increment local counts which are used to determine if the colors/textures are complete.

Other Example Compatible Implementations

[0095] Certain of the above-described system components 50 could be implemented as other than the home video game console configuration described above. For example, one could run graphics application or other software written for system 50 on a platform with a different configuration that emulates system 50 or is otherwise compatible with it. If the other platform can successfully emulate, simulate and/or provide some or all of the hardware and software resources of system 50, then the other platform will be able to successfully execute the software.

[0096] As one example, an emulator may provide a hardware and/or software configuration (platform) that is different from the hardware and/or software configuration (platform) of system 50. The emulator system might include software and/or hardware components that emulate or simulate some or all of hardware and/or software components of the system for which the application software was written. For example, the emulator system could comprise a general purpose digital computer such as a personal computer, which executes a software emulator program that simulates the hardware and/or firmware of system 50.

[0097] Some general purpose digital computers (e.g., IBM or MacIntosh personal computers and compatibles) are now equipped with 3D graphics cards that provide 3D graphics pipelines compliant with DirectX or other standard 3D graphics command APIs. They may also be equipped with stereophonic sound cards that provide high quality stereophonic sound based on a standard set of sound commands. Such multimedia-hardware-equipped personal computers running emulator software may have sufficient performance to approximate the graphics and sound performance of

system 50. Emulator software controls the hardware resources on the personal computer platform to simulate the processing. 3D graphics, sound, peripheral and other capabilities of the home video game console platform for which the game programmer wrote the game software.

[0098] Figure 21A illustrates an example overall emulation process using a host platform 1201, an emulator component 1303, and a game software executable binary image provided on a storage medium 62. Host 1201 may be a general or special purpose digital computing device such as, for example, a personal computer, a video game console, or any other platform with sufficient computing power. Emulator 1303 may be software and/or hardware that runs on host platform 1201, and provides a real-time conversion of commands, data and other information from storage medium 62 into a form that can be processed by host 1201. For example, emulator 1303 fetches "source" binary-image program instructions intended for execution by system 50 from storage medium 62 and converts these program instructions to a target format that can be executed or otherwise processed by host 1201.

[0099] As one example, in the case where the software is written for execution on a platform using an IBM PowerPC or other specific processor and the host 1201 is a personal computer using a different (e.g., Intel) processor, emulator 1303 fetches one or a sequence of binary-image program instructions from storage medium 62 and converts these program instructions to one or more equivalent Intel binary-image program instructions. The emulator 1303 also fetches and/or generates graphics commands and audio commands intended for processing by the graphics and audio processor 114, and converts these commands into a format or formats that can be processed by hardware and/or software graphics and audio processing resources available on host 1201. As one example, emulator 1303 may convert these commands into commands that can be processed by specific graphics and/or or sound hardware of the host 1201 (e.g., using standard DirectX, OpenGL and/or sound APIs).

[0100] An emulator 1303 used to provide some or all of the features of the video game system described above may also be provided with a graphic user interface (GUI) that simplifies or automates the selection of various options and screen modes for games run using the emulator. In one example, such an emulator 1303 may further include enhanced functionality as compared with the host platform for which the software was originally intended.

[0101] Figure 21B illustrates an emulation host system 1201 suitable for use with emulator 1303. System 1201 includes a processing unit 1203 and a system memory 1205. A system bus 1207 couples various system components including system memory 1205 to processing unit 1203. System bus 1207 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. System memory 1207 includes read only memory (ROM) 1252 and random access memory (RAM) 1254. A basic input/output system (BIOS) 1256, containing the basic routines that help to transfer information between elements within personal computer system 1201, such as during start-up, is stored in the ROM 1252. System 1201 further includes various drives and associated computer-readable media. A hard disk drive 1209 reads from and writes to a (typically fixed) magnetic hard disk 1211. An additional (possible optional) magnetic disk drive 1213 reads from and writes to a removable "floppy" or other magnetic disk 1215. An optical disk drive 1217 reads from and, in some configurations, writes to a removable optical disk 1219 such as a CD ROM or other optical media. Hard disk drive 1209 and optical disk drive 1217 are connected to system bus 1207 by a hard disk drive interface 1221 and an optical drive interface 1225, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules, game programs and other data for personal computer system 1201. In other configurations, other types of computer-readable media that can store data that is accessible by a computer (e.g., magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs) and the like) may also be used.

[0102] A number of program modules including emulator 1303 may be stored on the hard disk 1211, removable magnetic disk 1215, optical disk 1219 and/or the ROM 1252 and/or the RAM 1254 of system memory 1205. Such program modules may include an operating system providing graphics and sound APIs, one or more application programs, other program modules, program data and game data. A user may enter commands and information into personal computer system 1201 through input devices such as a keyboard 1227, pointing device 1229, microphones, joysticks, game controllers, satellite dishes, scanners, or the like. These and other input devices can be connected to processing unit 1203 through a serial port interface 1231 that is coupled to system bus 1207, but may be connected by other interfaces, such as a parallel port, game port Fire wire bus or a universal serial bus (USB). A monitor 1233 or other type of display device is also connected to system bus 1207 via an interface, such as a video adapter 1235.

[0103] System 1201 may also include a modem 1154 or other network interface means for establishing communications over a network 1152 such as the Internet. Modem 1154, which may be internal or external, is connected to system bus 123 via serial port interface 1231. A network interface 1156 may also be provided for allowing system 1201 to communicate with a remote computing device 1150 (e.g., another system 1201) via a local area network 1158 (or such communication may be via wide area network 1152 or other communications path such as dial-up or other communications means). System 1201 will typically include other peripheral output devices, such as printers and other standard peripheral devices.

[0104] In one example, video adapter 1235 may include a 3D graphics pipeline chip set providing fast 3D graphics

rendering in response to 3D graphics commands issued based on a standard 3D graphics application programmer interface such as Microsoft's DirectX 7.0 or other version. A set of stereo loudspeakers 1237 is also connected to system bus 1207 via a sound generating interface such as a conventional "sound card" providing hardware and embedded software support for generating high quality stereophonic sound based on sound commands provided by bus 1207. These hardware capabilities allow system 1201 to provide sufficient graphics and sound speed performance to play software stored in storage medium 62.

[0105] While the invention has been described in connection with what is presently considered to be the most practical and preferred embodiment, it is to be understood that the invention is not to be limited to the disclosed embodiment, but on the contrary, is intended to cover various modifications and equivalent arrangements included within the scope of the appended claims. For example, as will be appreciated, although a particular cartoon lighting example is disclosed herein, the invention is not to be limited to cartoon lighting but rather, encompasses a variety of photo-realistic and non-photorealistic, texture-based and non-texture-based applications and effects that may be accomplished using an achromatic lighting parameter.

Claims

1. In a graphics generating system (50) of the type that defines at least one per-vertex lighting function, a method of generating graphics images **characterized by**:

   using the lighting function to calculate at least one parameter other than color or opacity to be used in a subsequent modification of a color or opacity; and
   rendering an image based at least in part on the calculated parameter.

2. The method as in claim 1 further **characterized in that** the parameter comprises at least one texture coordinate.

3. The method as in any one of the preceding claims further **characterized in that** the lighting function calculates said at least one parameter based at least in part on distance attenuation between a vertex and a light source.

4. The method as in any one of the preceding claims further **characterized in that** said lighting function calculates said at least one parameter based at least in part on the angle between the light source to vertex direction and the surface normal

5. The method as in any one of the preceding claims further **characterized in that** said lighting function calculates said at least one parameter based at least in part on a vertex vector position.

6. The method as in any one of the preceding claims further **characterized in that** said lighting function defines a function based upon surface position, surface orientation and the position of at least one light.

7. The method as in any one of the preceding claims further **characterized in that** said lighting function calculates a negative parameter for back lighting.

8. The method as in any one of the preceding claims further **characterized in that** said lighting function can use diffuse or specular lighting.

9. The method as in any one of the preceding claims further **characterized in that** the rendering step comprises generating a texture for a surface based at least in part on said parameter.

10. The method as in any one of the preceding claims further **characterized in that** said rendering step comprises texture mapping using a one-dimensional texture indexed by said at least one parameter.

11. The method of any of the preceding claims further **characterized in that** the method further includes:

    using a lighting function to generate at least one color or opacity value;
    converting said color opacity value to at least one texture coordinate;
    using said texture coordinate in at least one texture mapping operation; and
    using the results of said texture mapping operation to modify the color or opacity of at least one visible surface in a dynamically generated cartoon image.

SDOCID: <EP_____1189173A2 I_>

12. The method as in claim 11 further **characterized in that** said texture mapping operation comprises mapping a one-dimensional texture in response to said texture coordinate.

13. The method as in any one of the preceding claims 11-12 further **characterized in that** said texture mapping comprises mapping brush strokes to dynamic lighting computation outputs.

14. The method as in any one of the preceding claims 11-13 further **characterized in that** said texture coordinate can range negatively to define cartoon backlighting.

15. The method as in any one of the preceding claims further **characterized in that** said lighting function defines a first texture coordinate for said texture mapping and a second texture coordinate for said texture mapping is derived from a source other than said lighting function.

16. The method as in any one of the preceding claims further **characterized in that** said lighting function comprises a diffuse lighting function.

17. The method as in any one of the preceding claims further **characterized in that** said lighting function comprises a specular lighting function.

18. The method as in any one of the preceding claims further **characterized in that** said lighting function comprises a distance-attenuated lighting function.

19. The method as in any one of the preceding claims further **characterized in that** said lighting function comprises a spotlight lighting function.

20. The method as in any one of the preceding claims further **characterized in that** the generated parameters can be either host computed or computed using the lighting function, as required.

21. The method as in any one of the preceding claims further **characterized in that** one generated parameter is used to select a texel within a 1D texture and a second parameter is used to select which 1D texture to use.

22. In a graphics system (50) comprising a lighting data pipeline (300) comprising at least first and second channels, said lighting data pipeline (300) receiving per-vertex information and lighting definitions, and calculating lit vertex parameters in response to said per-vertex and lighting definitions;
the system (50) further **characterized by**:

a texture unit (500) coupled to said lighting data pipeline (300), said texture unit (500) performing at least one texture mapping operation using at least one of said lighting data pipeline (300) outputs as at least one texture coordinate; and
a blender (700) coupled to said lighting data pipeline (300) and said texture unit (150), said blender (700) blending chromatic and/or opacity information obtained from said lighting data pipeline (300) with texels obtained from said texture unit (500)to provide a rendered image.

23. The system as in claim 22 further **characterized in that** the lighting data pipeline outputs comprise at least one texture coordinate.

24. The system as in any one of the preceding claims 22-23 further **characterized in that** the lighting data pipeline output is calculated based at least in part on distance attenuation between a vertex and a light source.

25. The system as in any one of the preceding claims 22-24 further **characterized in that** said lighting data pipeline output is calculated based at least in part on the angle between the light source to vertex direction and the surface normal.

26. The system as in any one of the preceding claims 22-25 further **characterized in that** said lighting data pipeline output is calculated based at least in part on a vertex vector position.

27. The system as in any one of the preceding claims 22-26 further **characterized in that** said lighting data pipeline output defines a function based upon surface position, surface orientation and the position of at least one light.

28. The system as in any one of the preceding claims 22-27 further **characterized in that** said lighting data pipeline output is calculated in the form of a negative parameter for back lighting.

29. The system as in any one of the preceding claims 22-28 further **characterized in that** said lighting data pipeline outputs can be for diffuse or specular lighting.

30. The system as in any one of the preceding claims 22-29 further **characterized in that** said texture unit (150) uses a one-dimensional texture indexed by said at least one parameter.

31. The system as in any one of the preceding claims 22-30 further **characterized in that** said texture unit (500) maps brush strokes to dynamic lighting computation outputs.

32. The system as in any one of the preceding claims 22-31 further **characterized in that** said texture coordinate can range negatively to define cartoon backlighting.

33. The system as in any one of the preceding claims 22-32 further **characterized in that** said lighting output defines a first texture coordinate for said texture mapping and a second texture coordinate for said texture mapper (500) is derived from a source other than said lighting output.

34. The system as in any one of the preceding claims 22-33 further **characterized in that** said lighting output comprises a diffuse lighting function.

35. The system as in any one of the preceding claims 22-34 further **characterized in that** said lighting output comprises a specular lighting function.

36. The system as in any one of the preceding claims 22-35 further **characterized in that** said lighting output comprises a distance-attenuated lighting function.

37. The system as in any one of the preceding claims 22-36 further **characterized in that** said lighting output comprises a spotlight lighting function.

SDOCID: <EP_____1189173A2_I_>

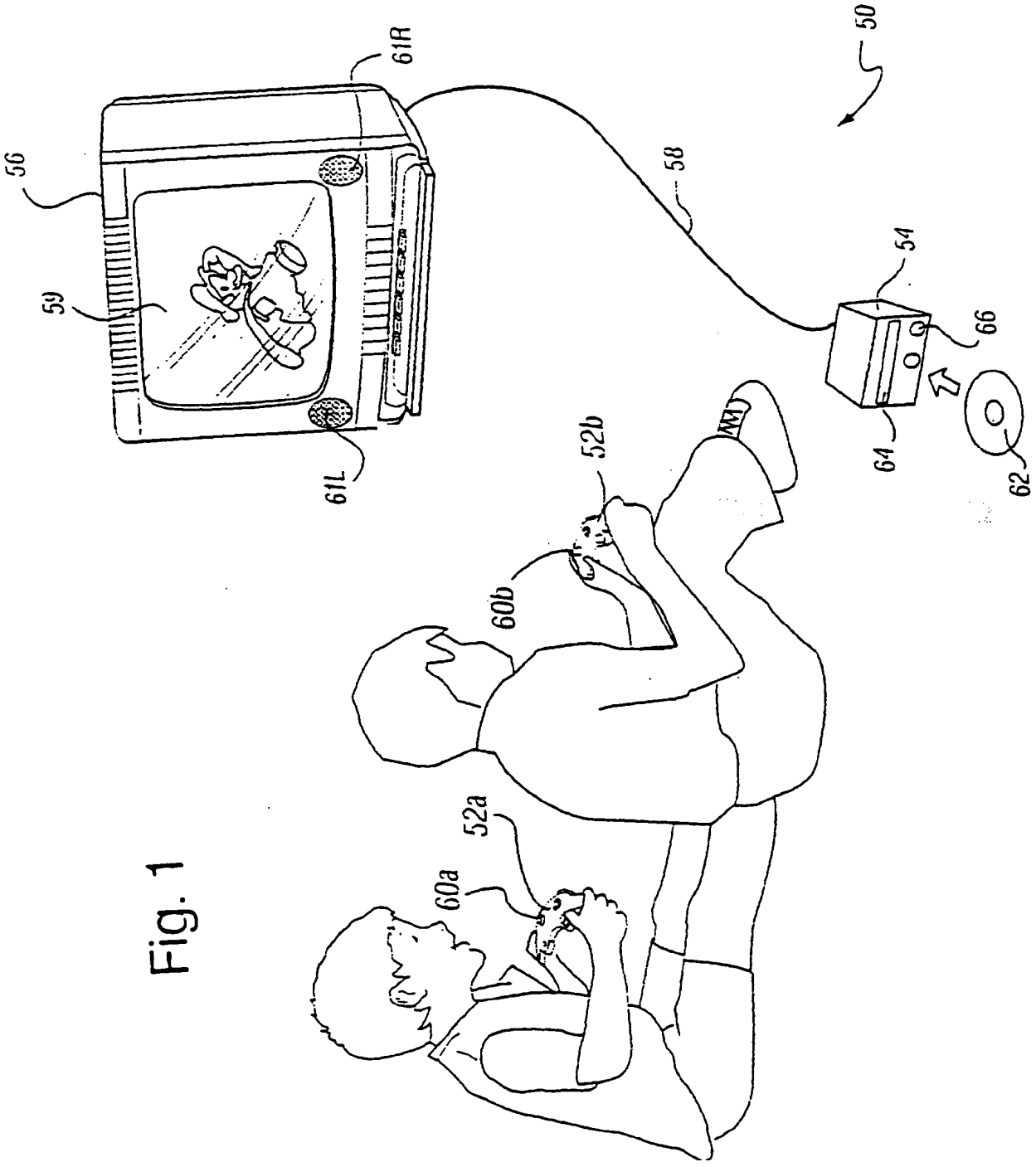


Fig. 1

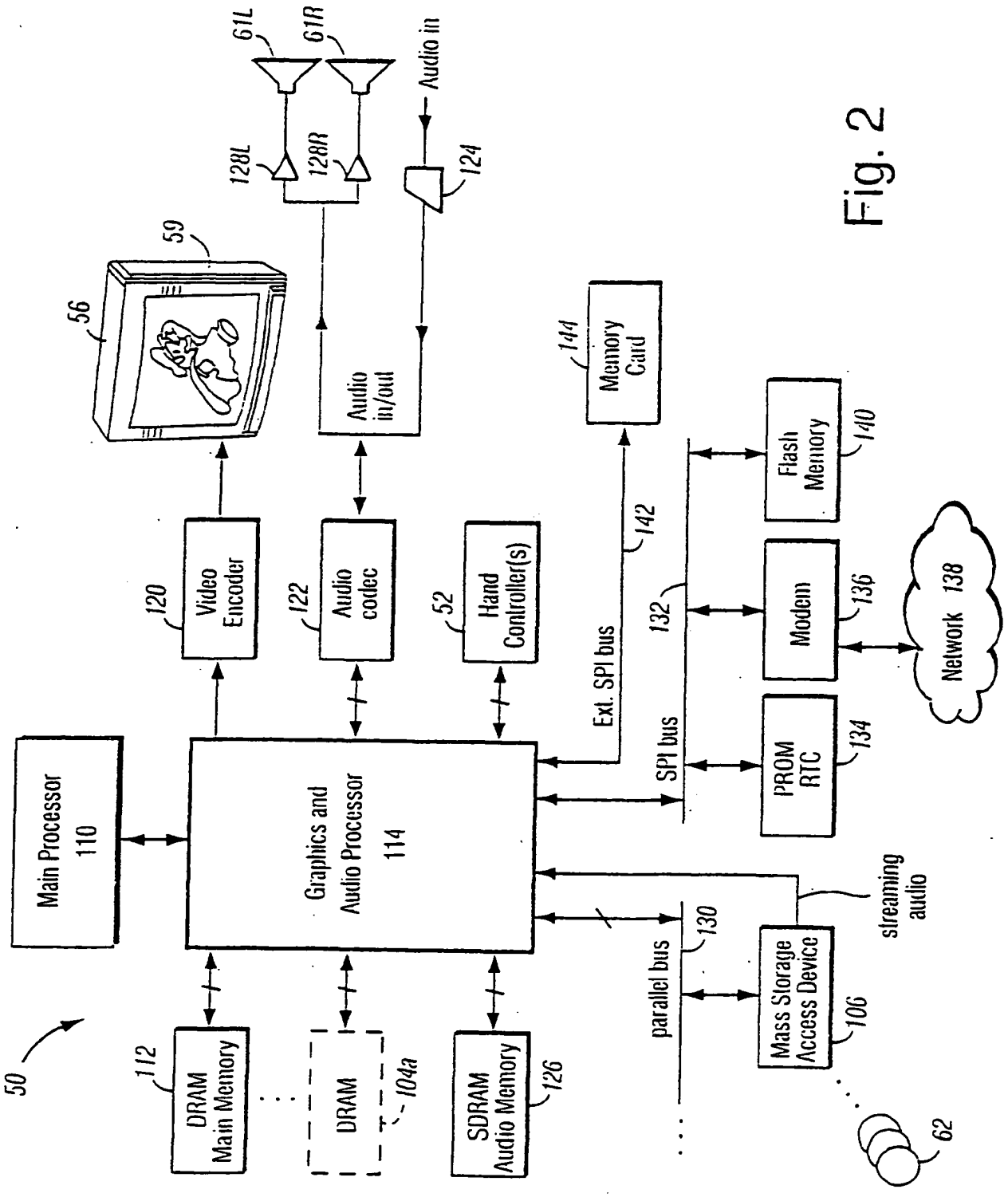


Fig. 2

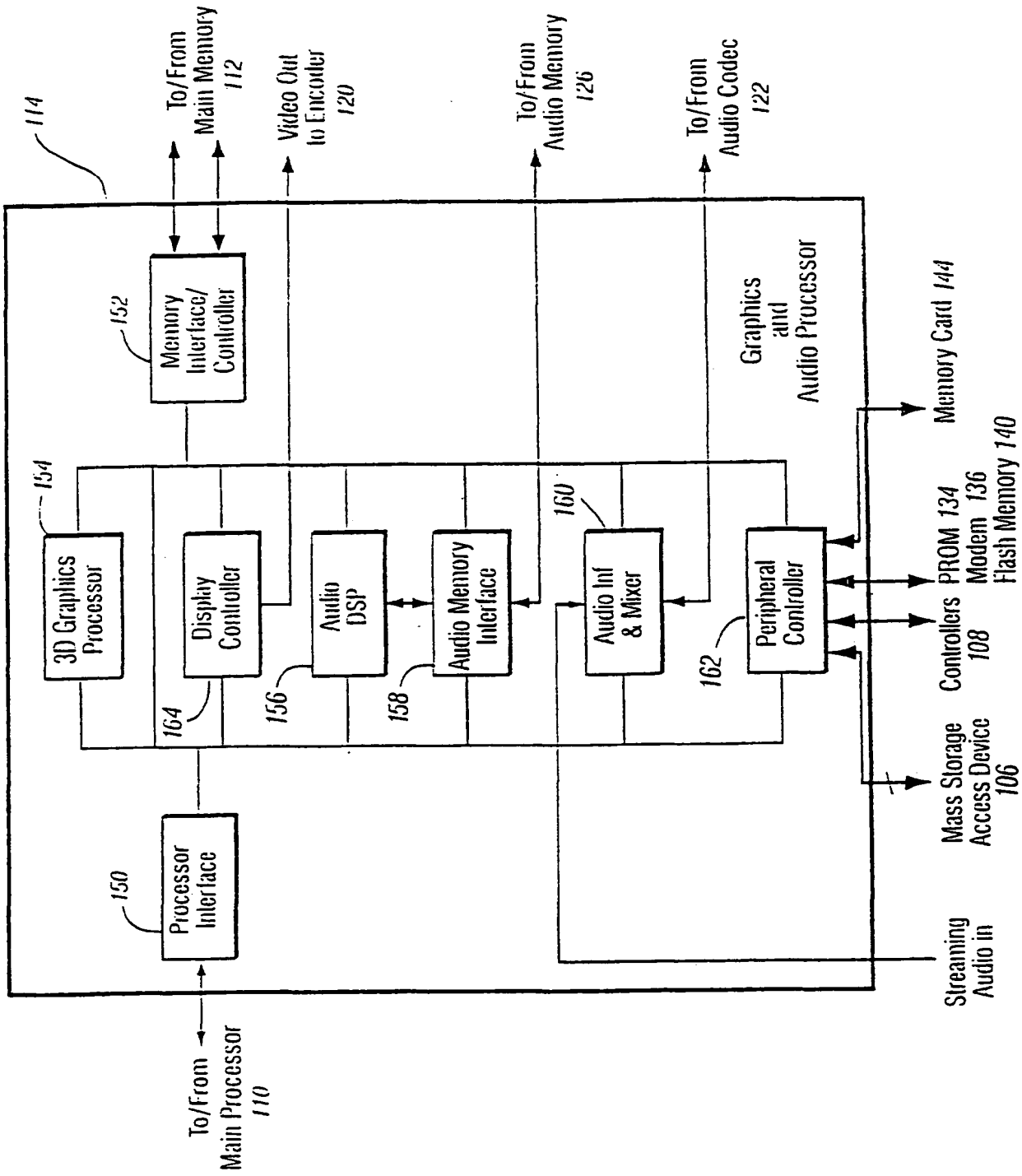


Fig. 3

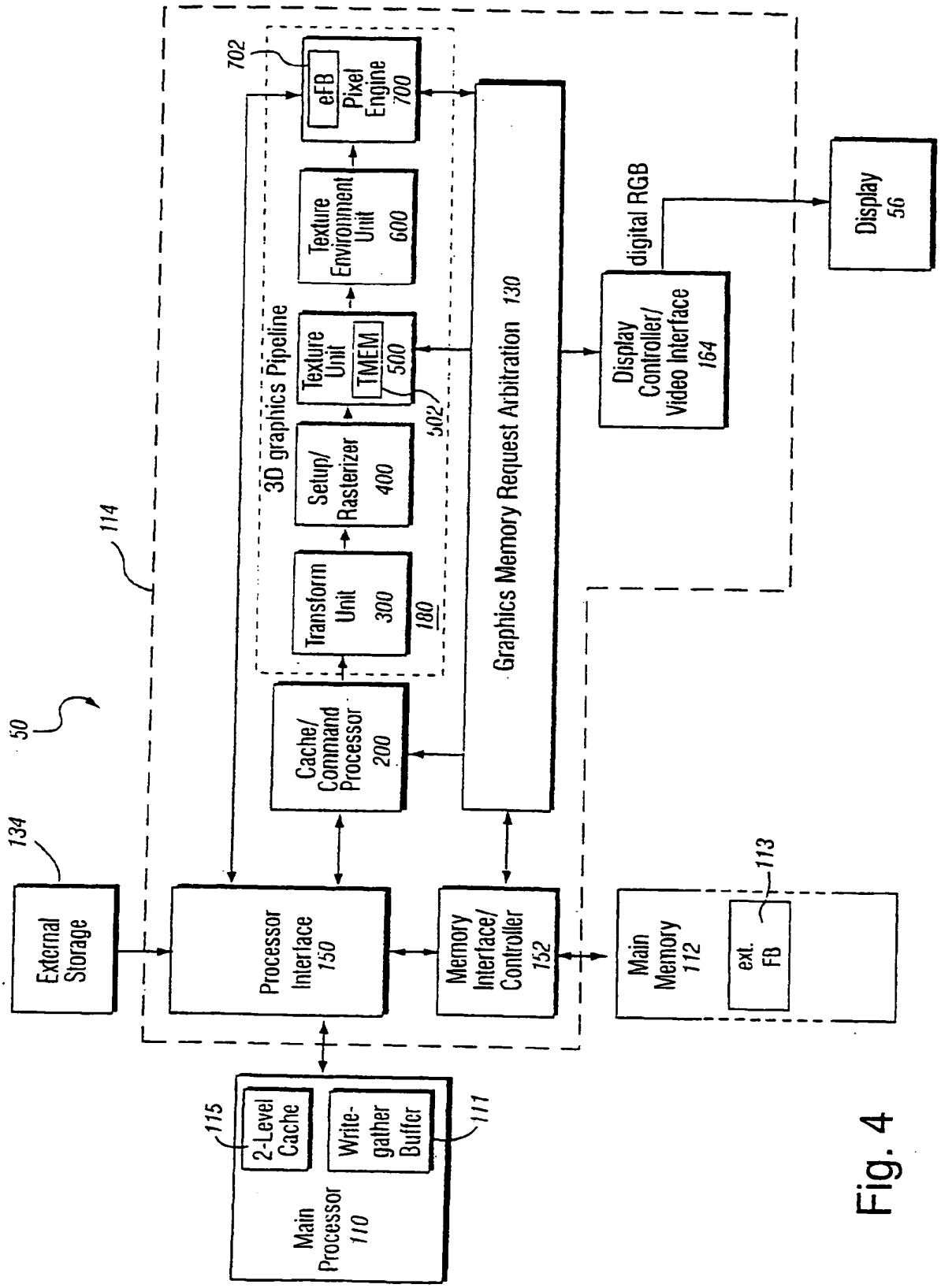


Fig. 4

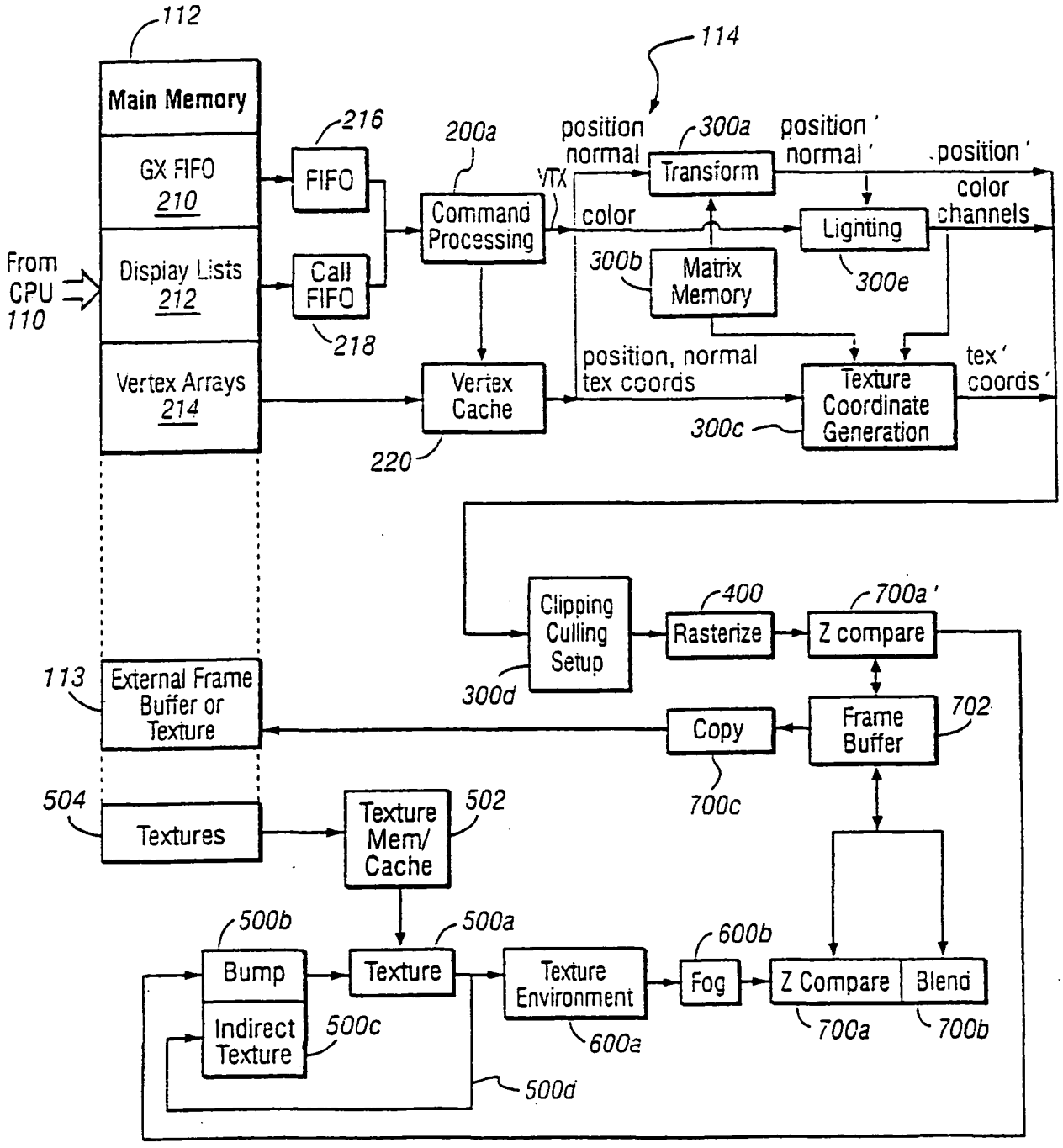


Fig. 5 EXAMPLE GRAPHICS PROCESSOR FLOW

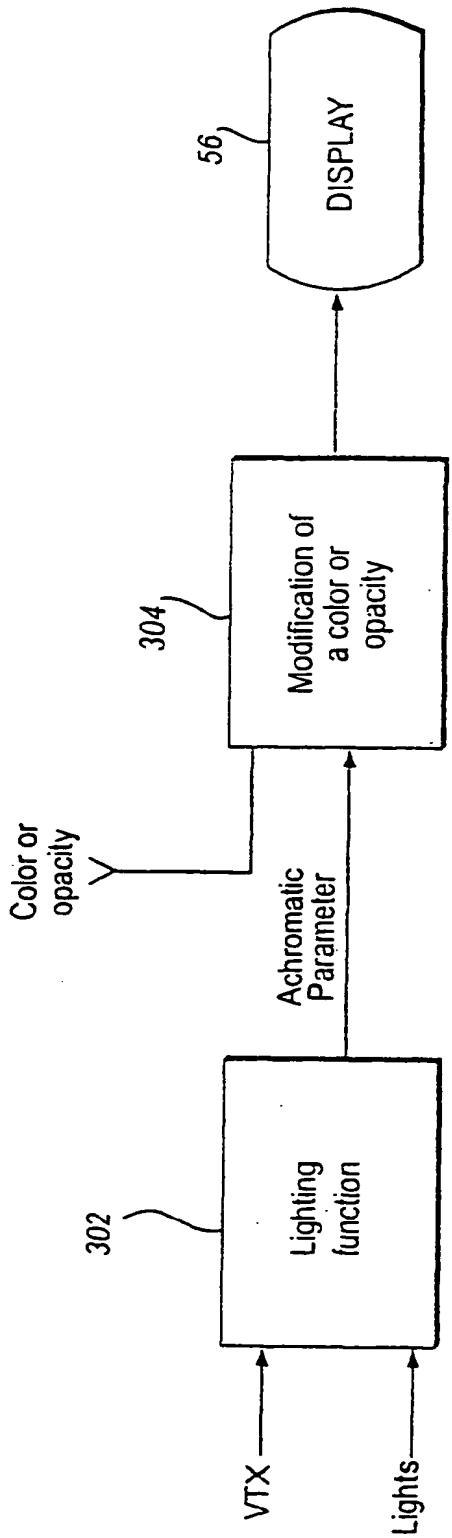


Fig. 6 EXAMPLE IMAGE GENERATING PROCESS

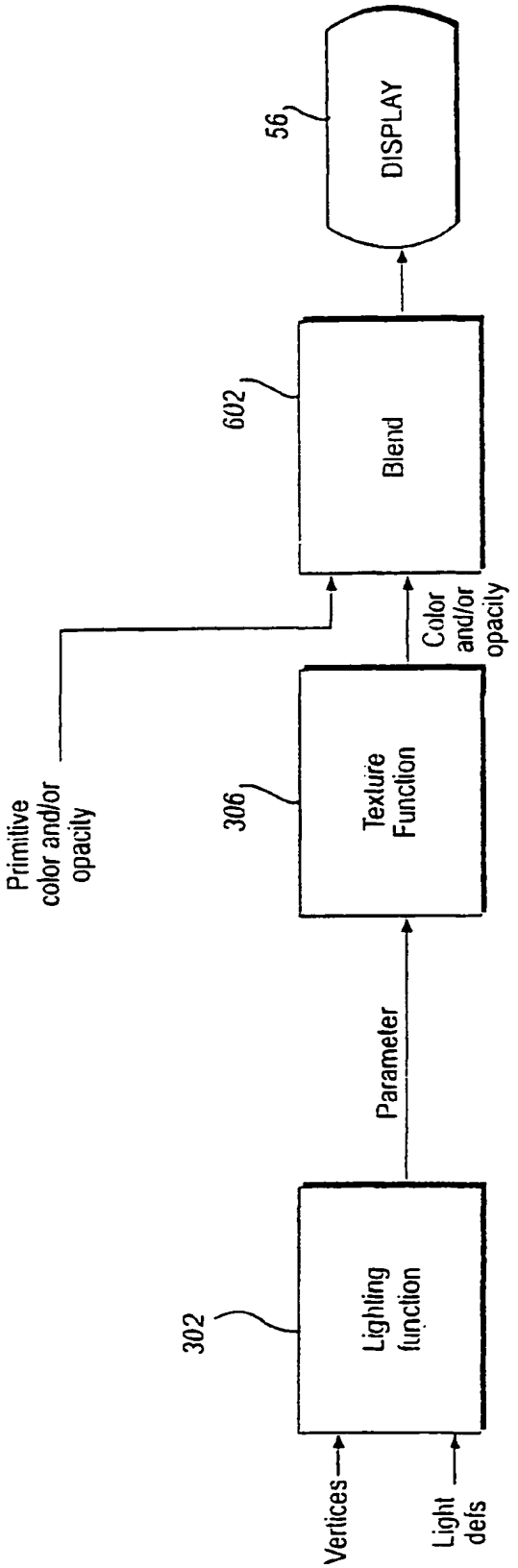


Fig. 7 EXAMPLE TEXTURING PROCESS BASED ON ACHROMATIC LIGHTING PARAMETER

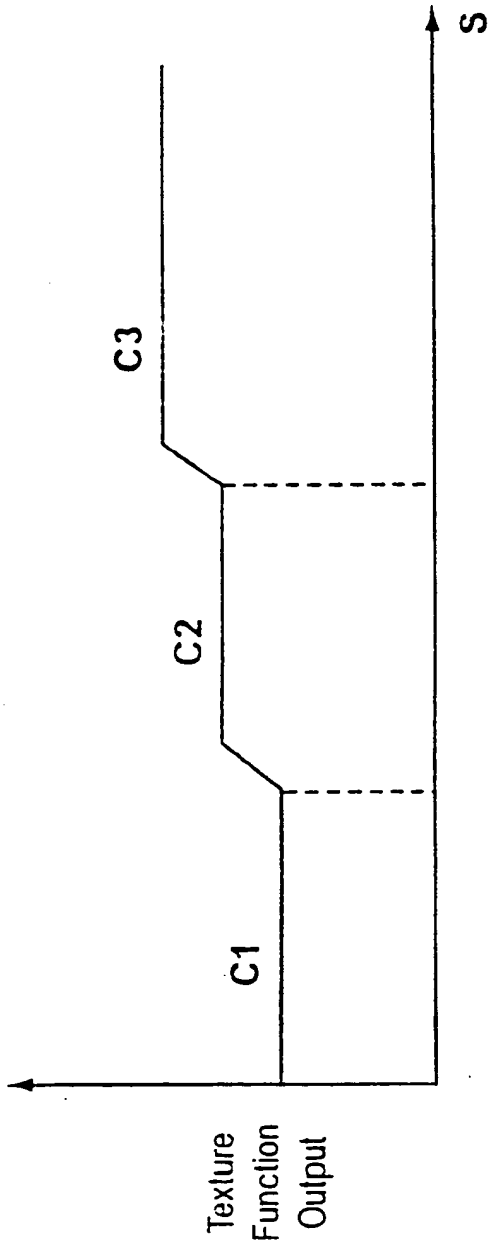


Fig. 8A EXAMPLE TEXTURE FUNCTION
Output = f(S)

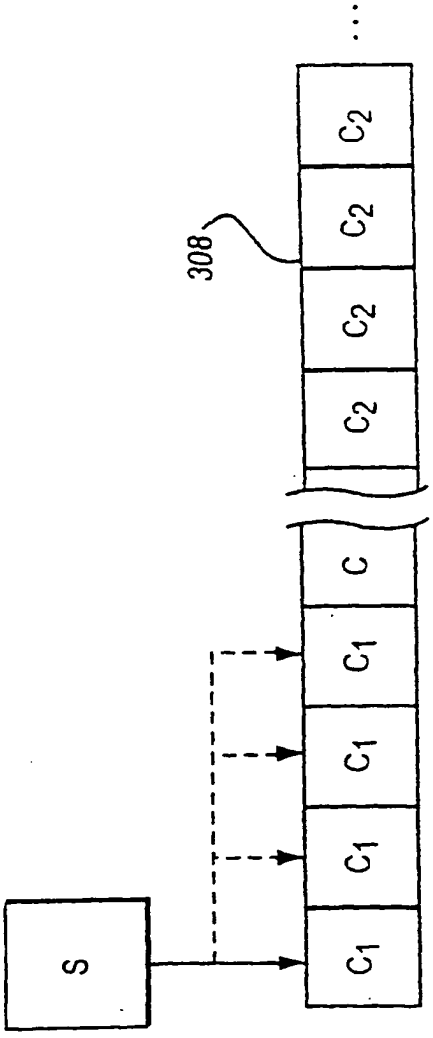


Fig. 8B EXAMPLE 1-D TEXTURE MAPPING

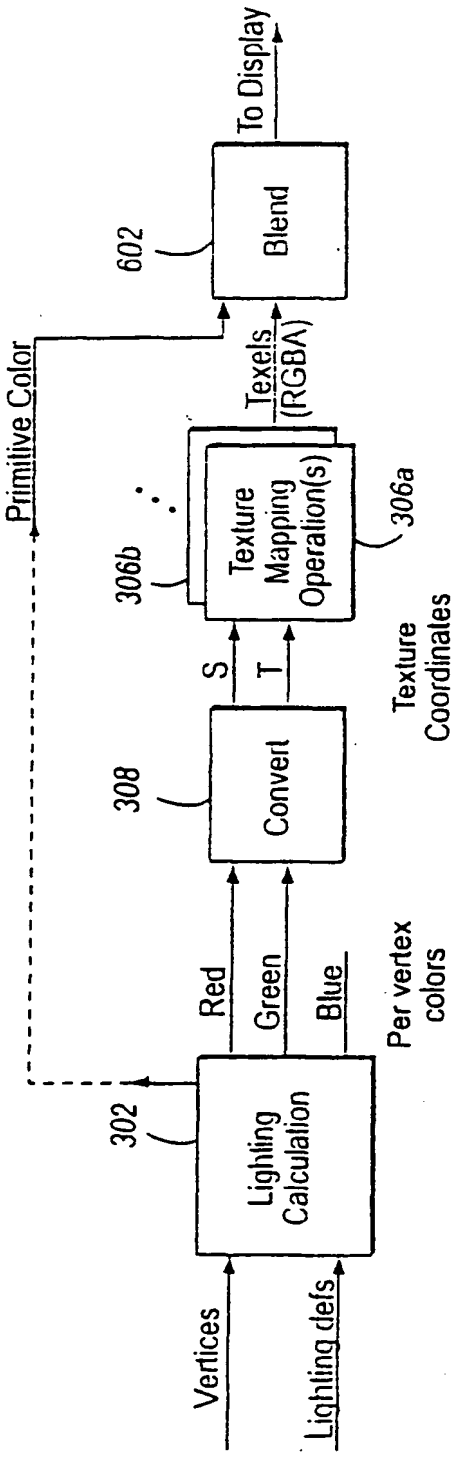


Fig. 9 EXAMPLE TEXTURE MAPPING

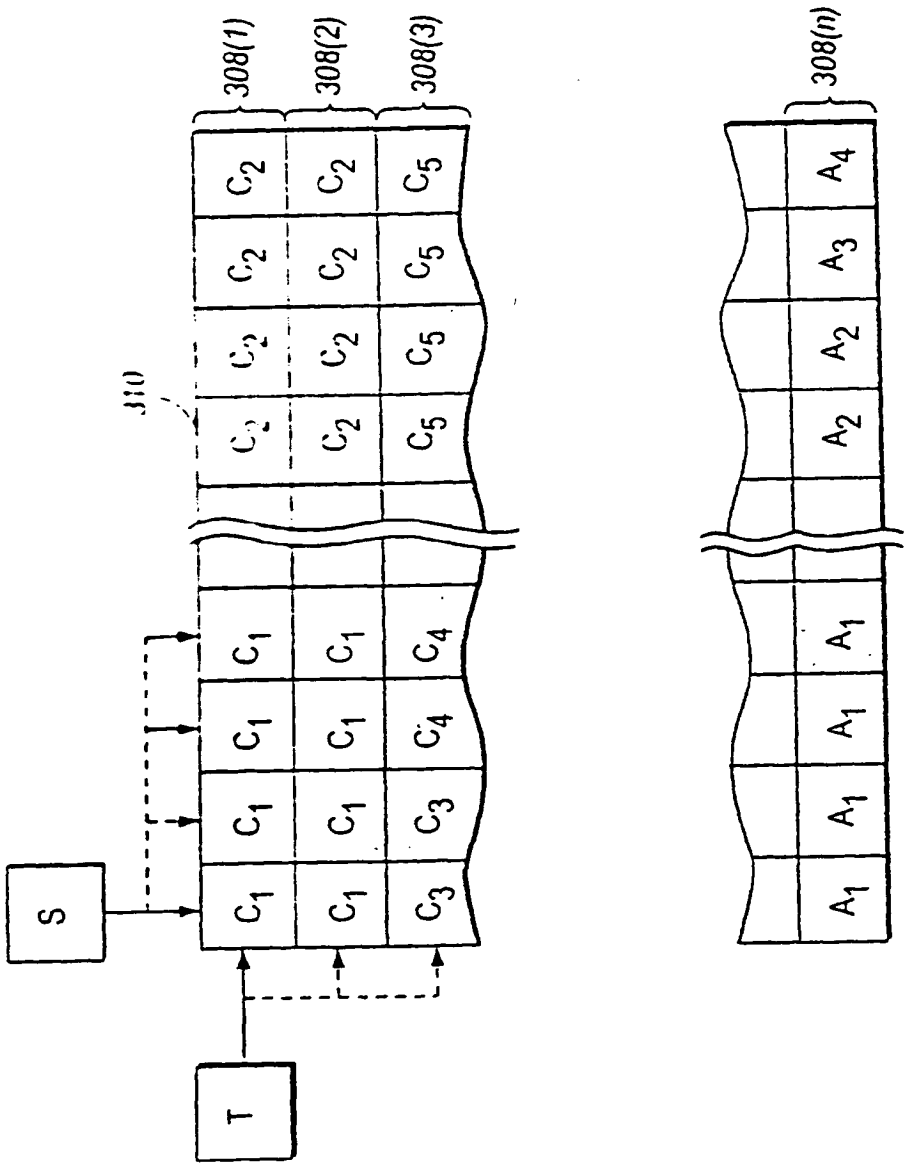


Fig. 10A EXAMPLE 2-D TEXTURE MAPPING

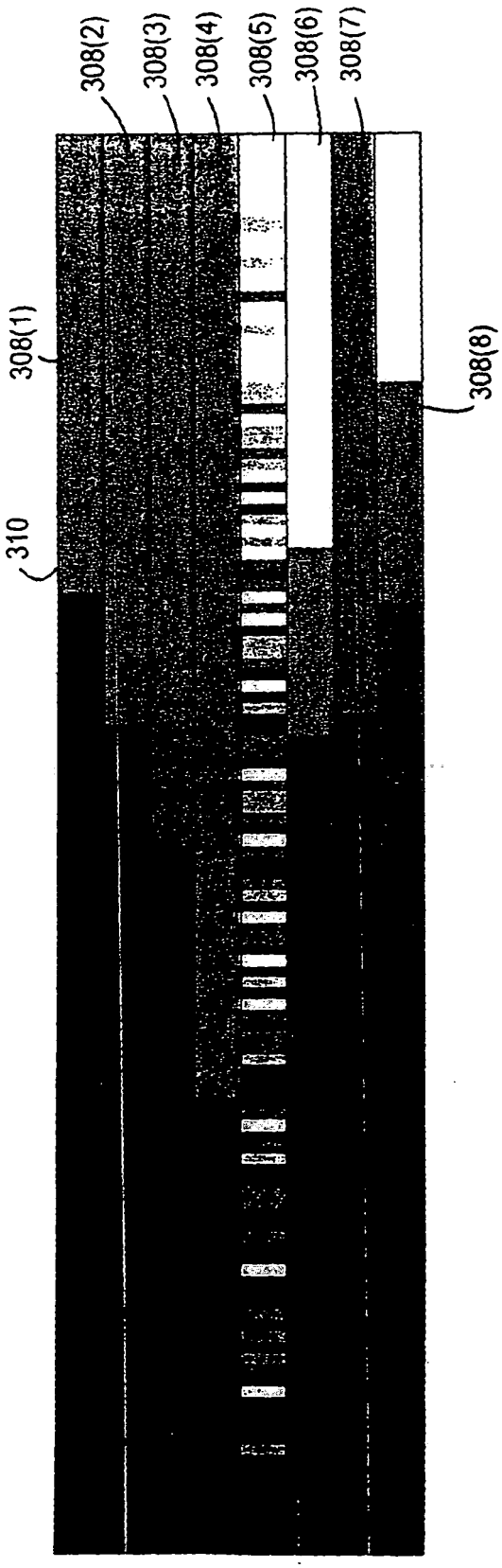


Fig. 10B
EXAMPLE 2D TEXTURE

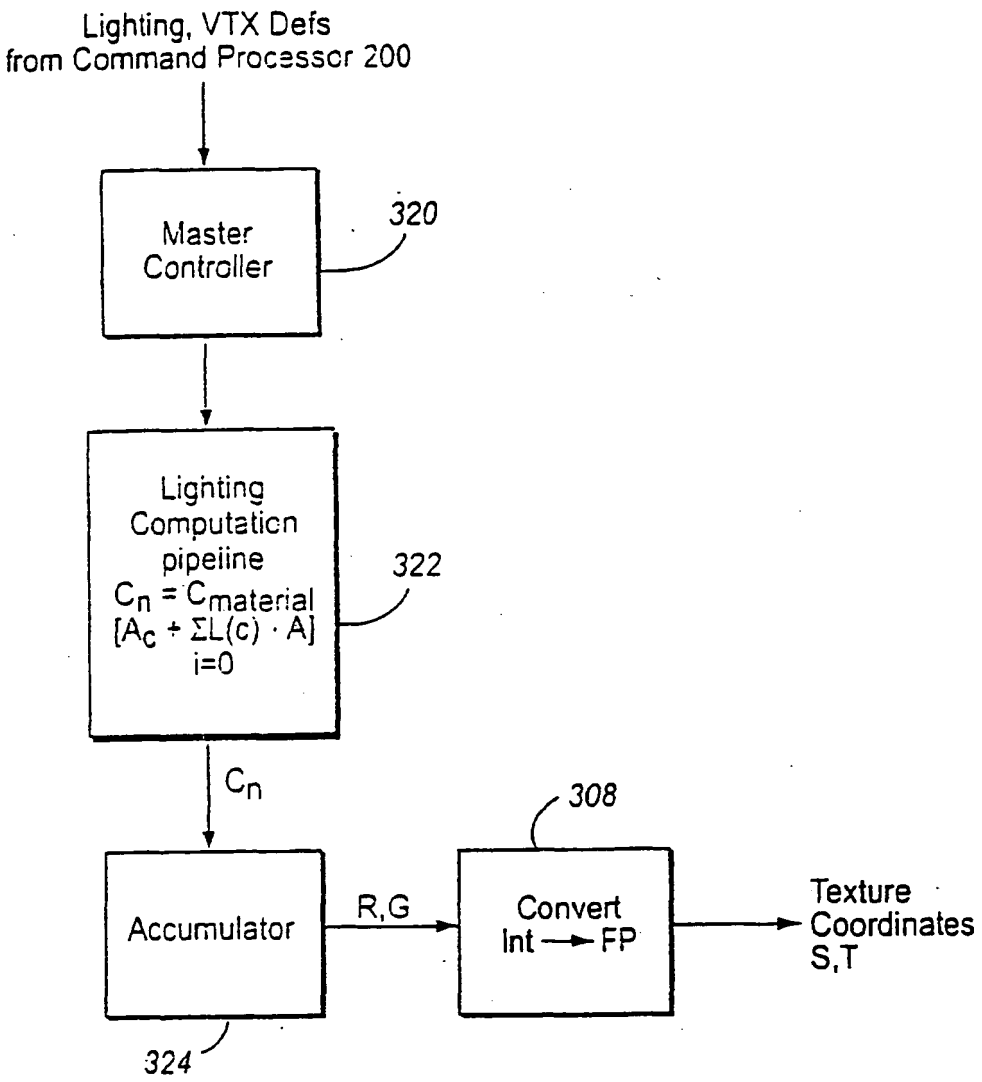


Fig. 11

EXAMPLE LIGHTING PIPELINE IMPLEMENTATION

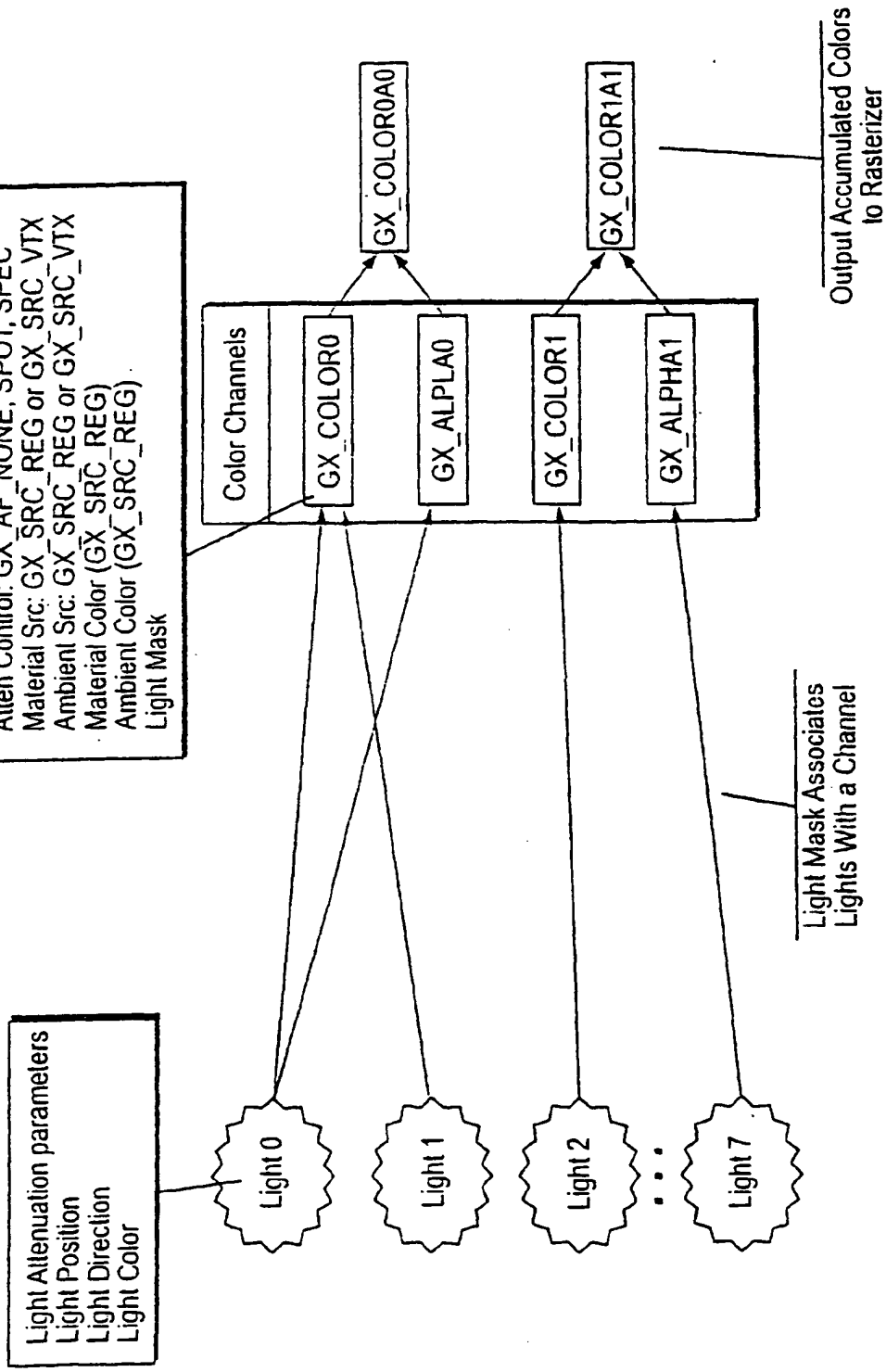


Fig. 12

ASSOCIATING LIGHTS WITH COLOR CHANNELS

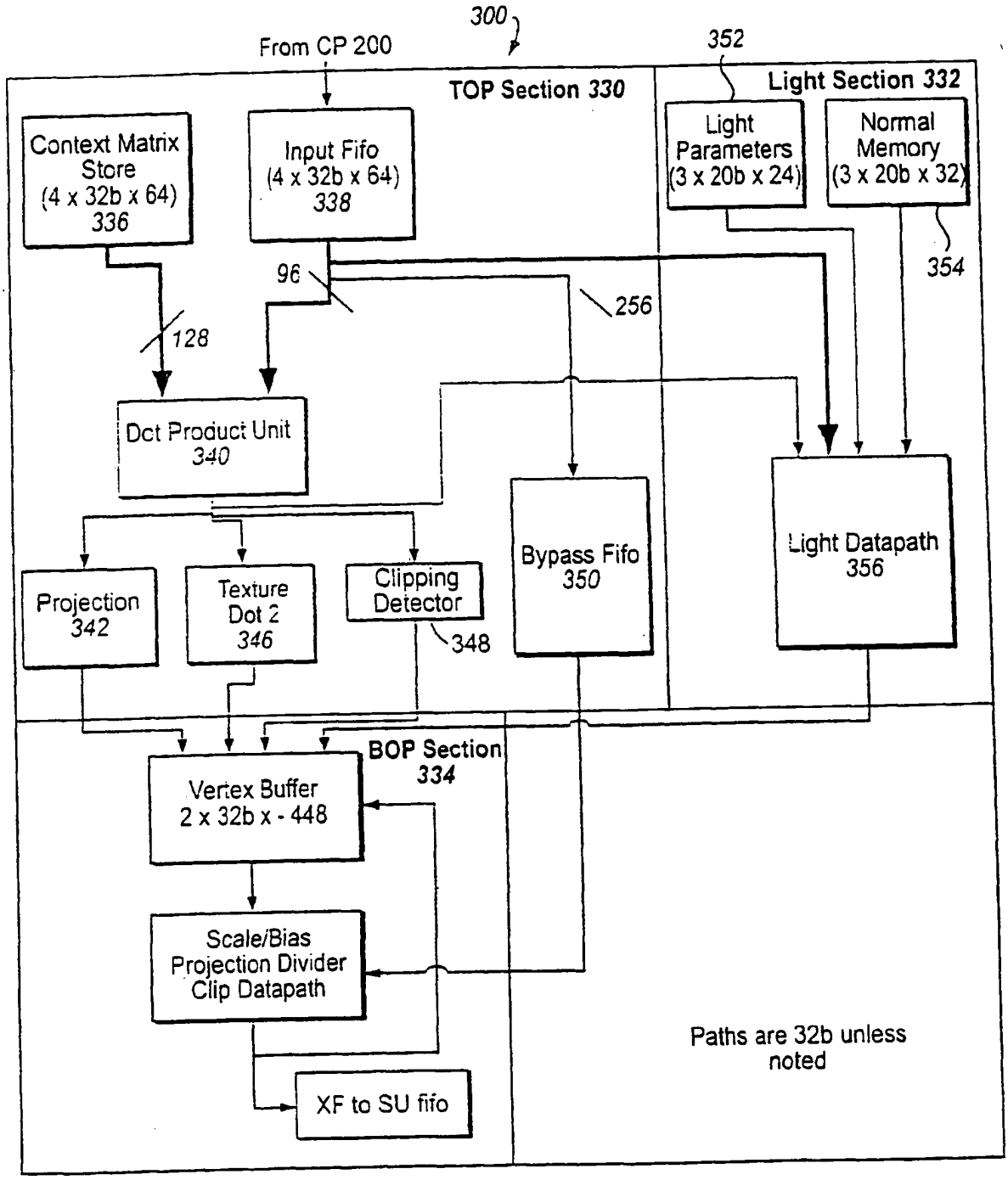


Fig. 13

EXAMPLE TRANSFORM UNIT IMPLEMENTATION

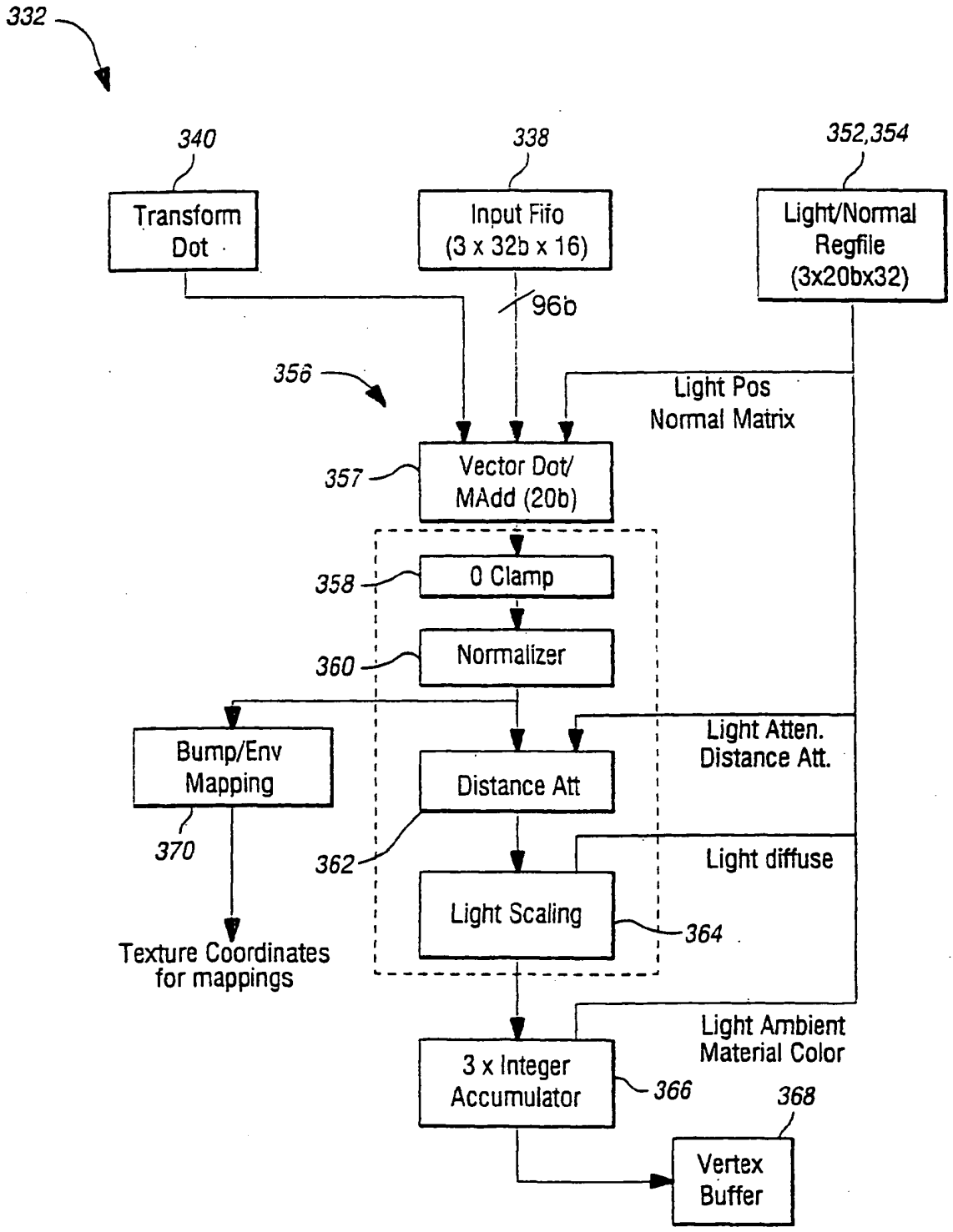


Fig. 14

Example Lighting Pipeline Implementation

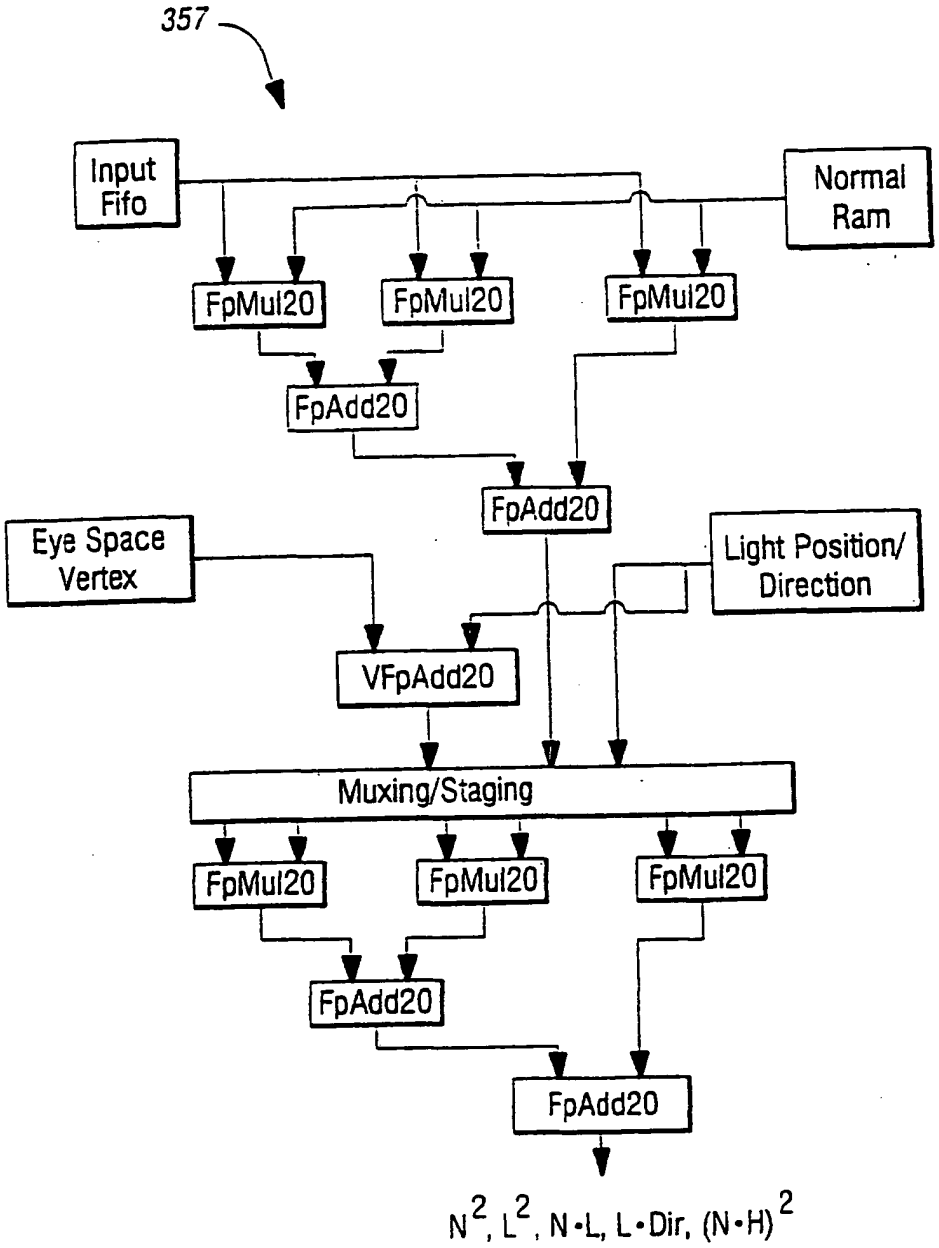


Fig. 15
Example Vector Dot/Add Unit Implementation

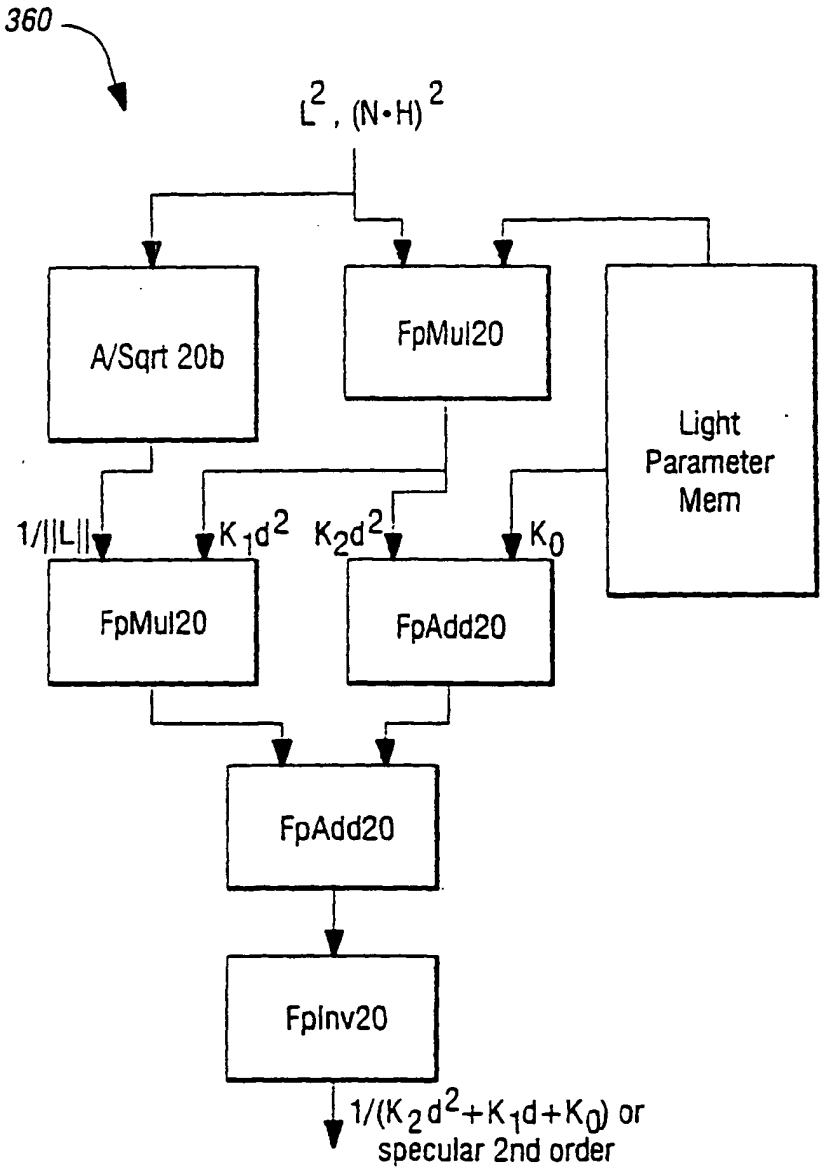


Fig. 16
Example Normalize Implementation

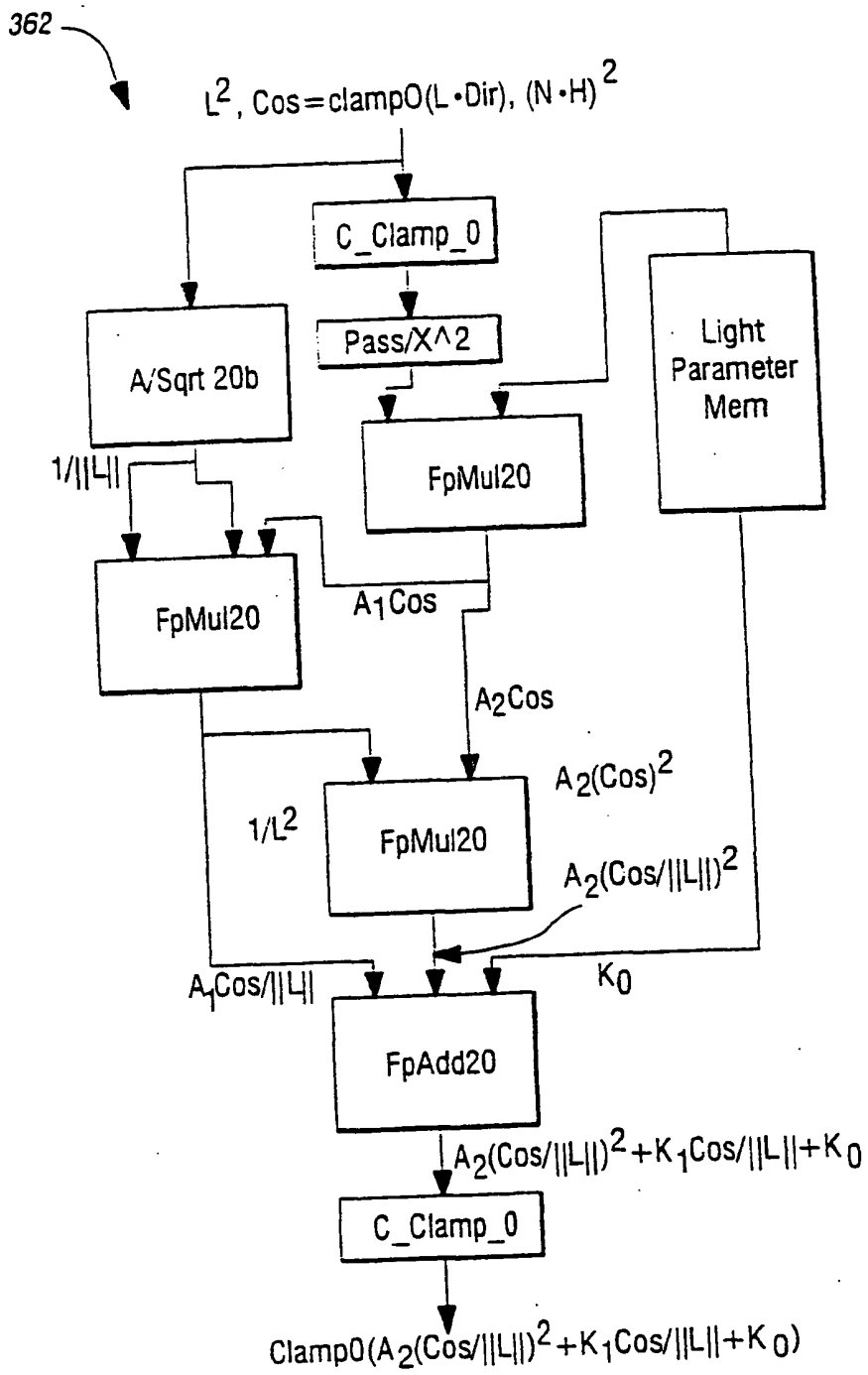


Fig. 17

Example Distance Attenuator Implementation

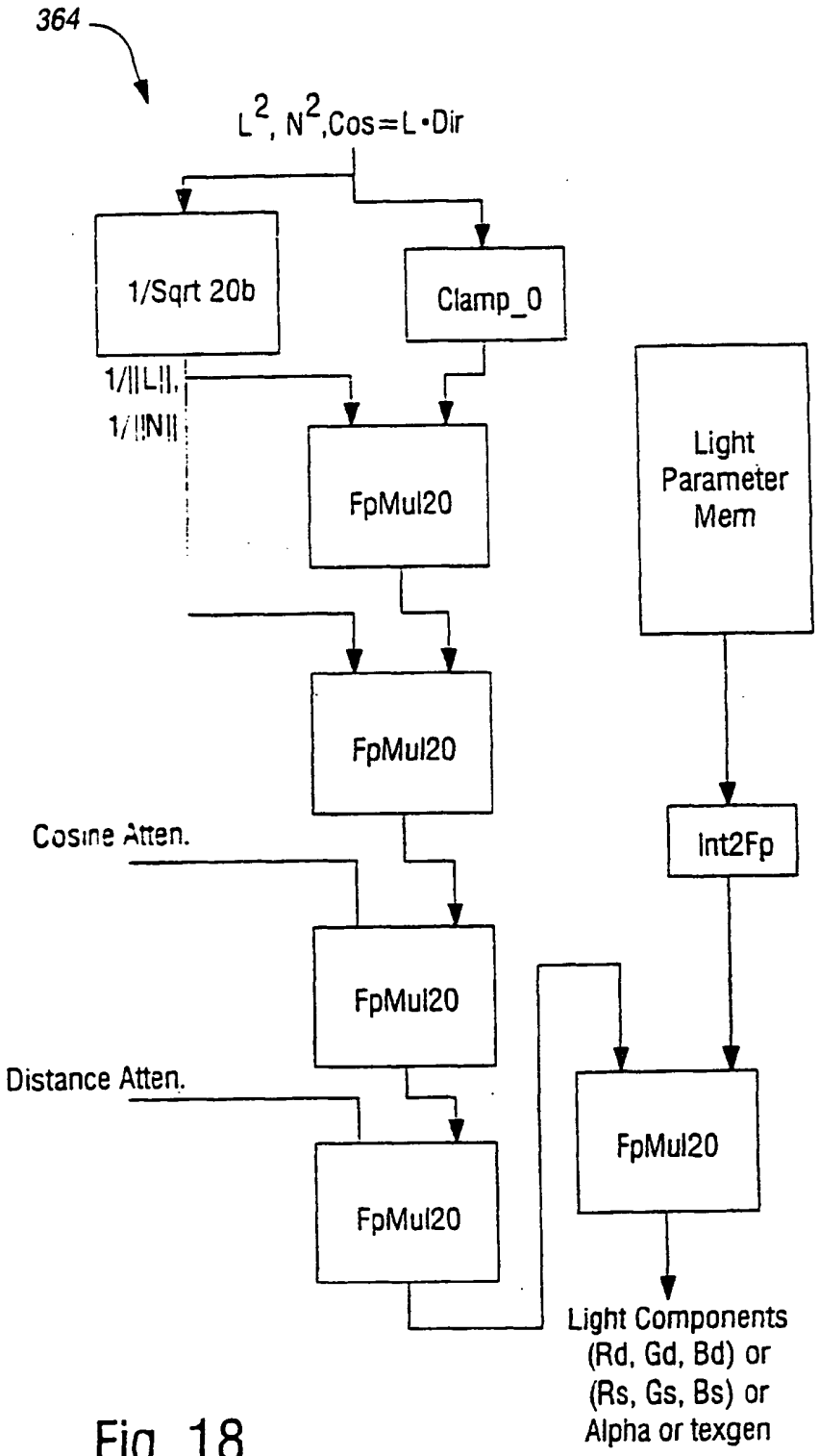
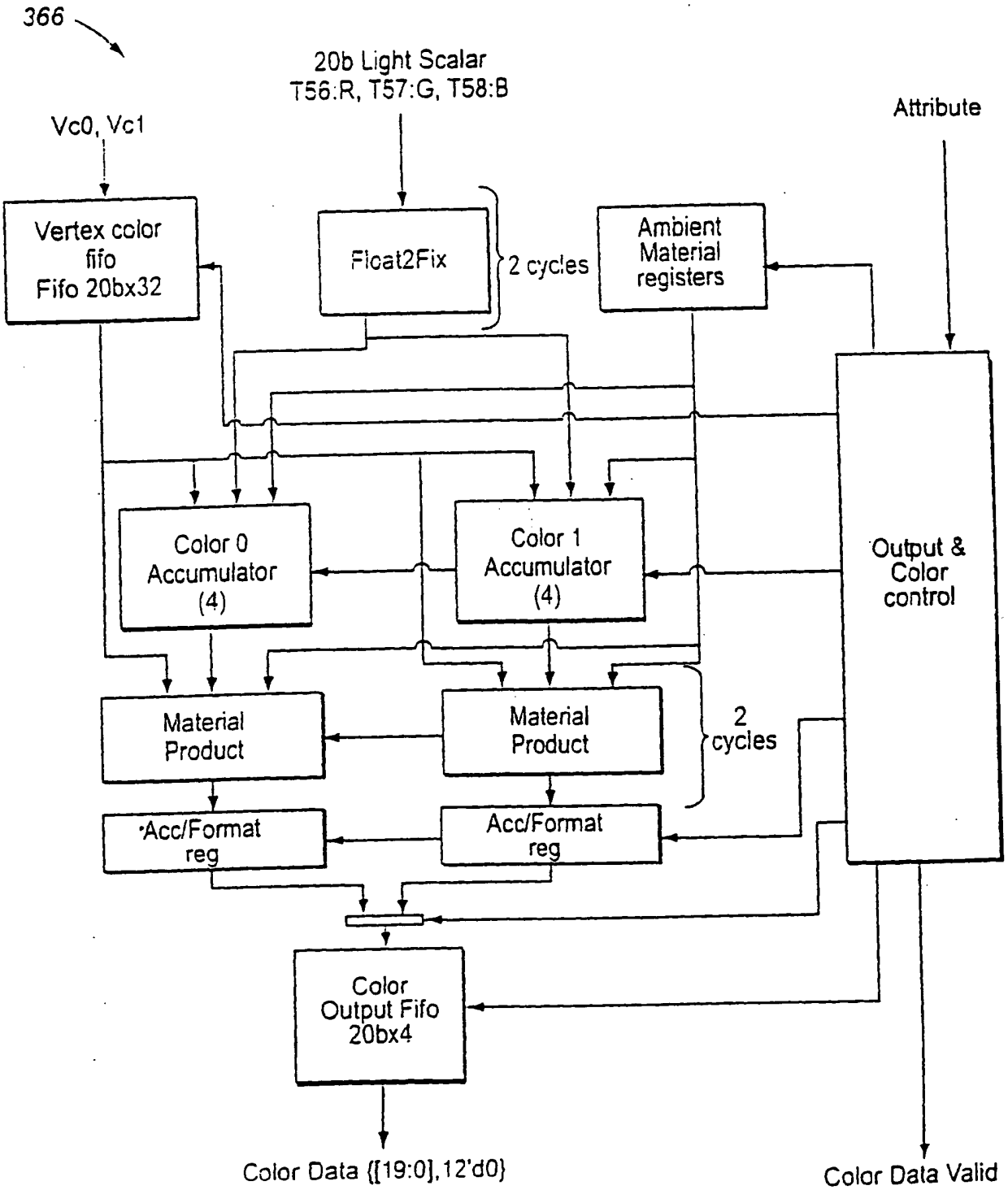


Fig. 18
Example Light Scaler Implementation

Fig. 19 EXAMPLE INTEGER ACCUMULATOR IMPLEMENTATION



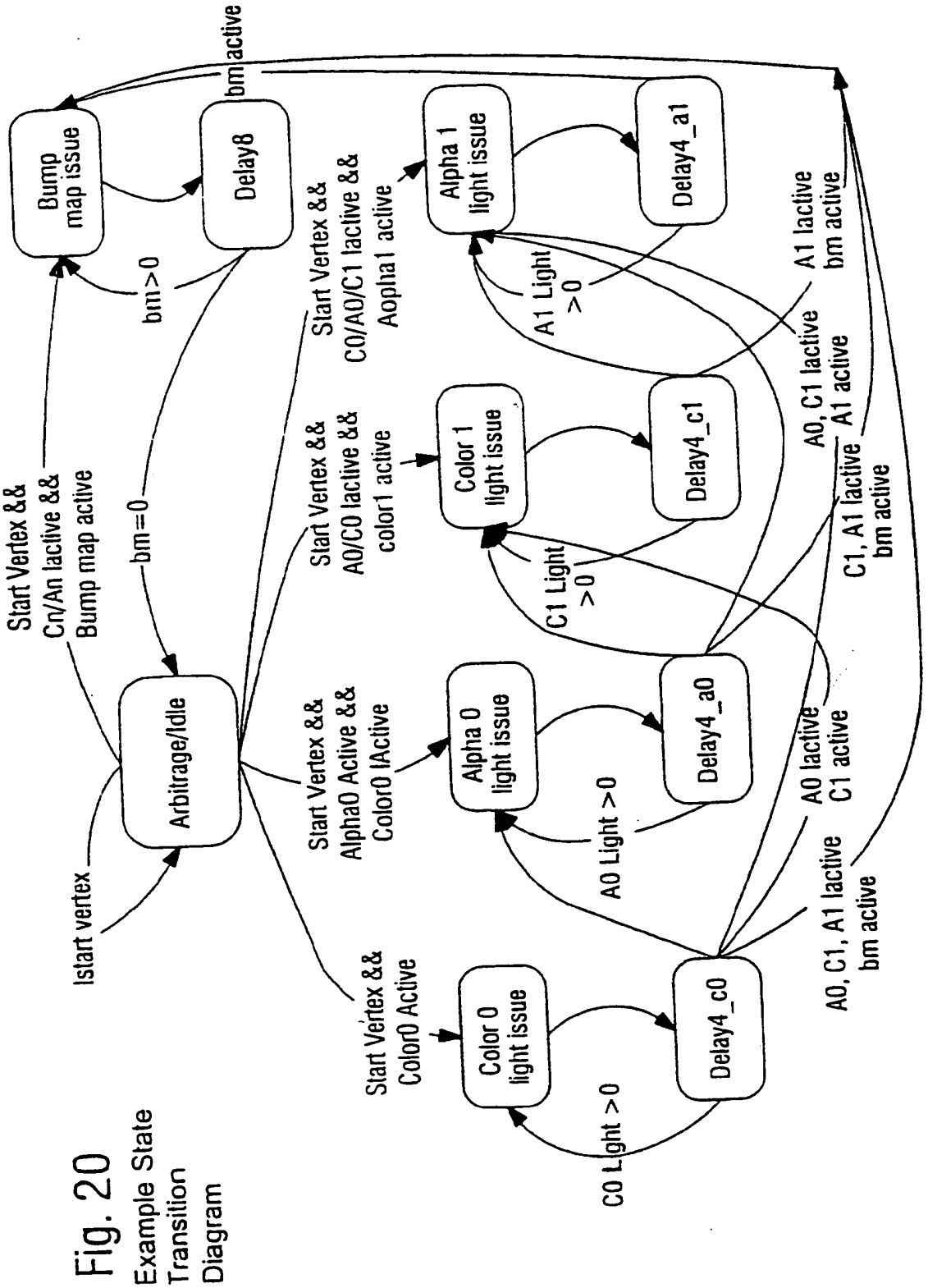


Fig. 20
Example State Transition Diagram

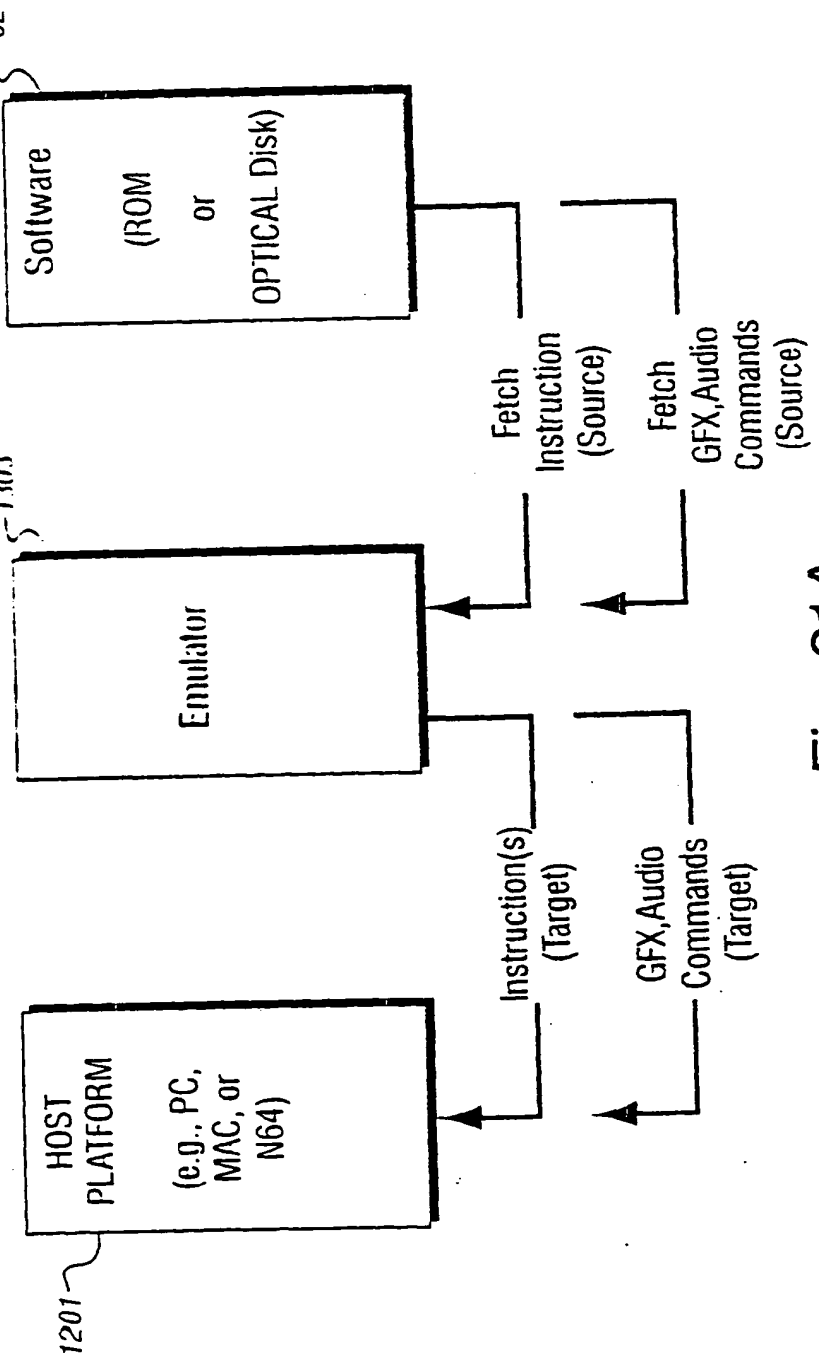


Fig. 21A

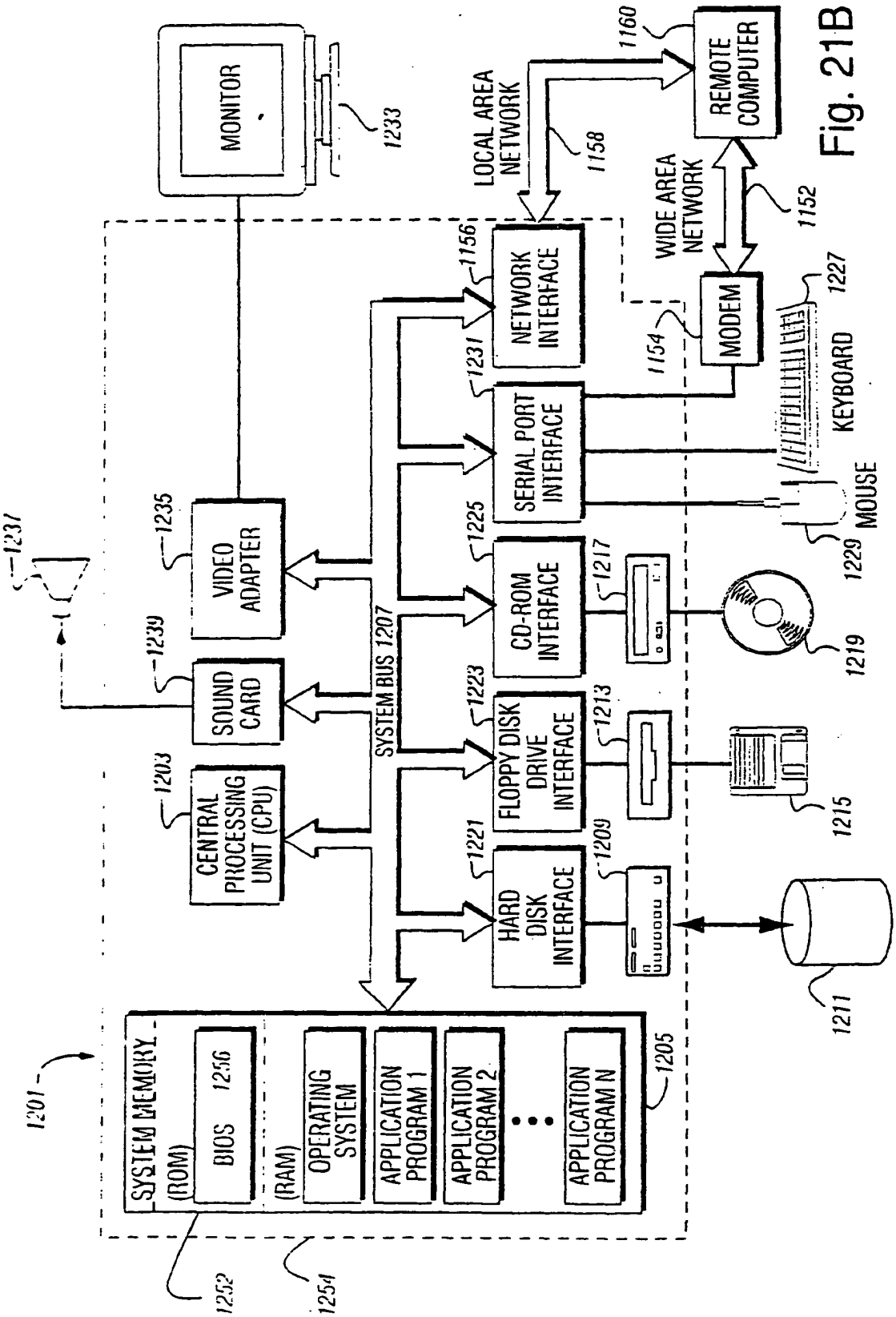


Fig. 21B

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 189 173 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
25.02.2004 Bulletin 2004/09

(43) Date of publication A2:
20.03.2002 Bulletin 2002/12

(21) Application number: **01307190.7**

(22) Date of filing: **23.08.2001**

(51) Int Cl.[7]: **G06T 15/50**, G06T 11/00, G06T 15/20, G06T 15/00

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE TR
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **23.08.2000 US 227007**
28.11.2000 US 726216

(71) Applicant: **Nintendo Co., Limited**
Minami-ku, Kyoto 601-8501 (JP)

(72) Inventors:
- **Drebin, Robert A.**
  California 94301 (US)
- **Yasumoto, Yoshitaka**
  Osaka 573-1111 (US)
- **Hollis, Martin**
  Cambridge CB2 1RQ (GB)
- **Demers, Eric**
  c/o ATI Technologies Systems Corp.
  Santa Clara CA 95051 (US)

(74) Representative: **Allman, Peter John et al**
MARKS & CLERK,
Sussex House,
83-85 Mosley Street
Manchester M2 3LG (GB)

(54) Achromatic lighting in a graphics system and method

(57) A graphics system (50) including a custom graphics and audio processor (114) produces exciting 2D and 3D graphics and surround sound. The system (50) includes a graphics and audio processor (114) including a 3D graphics pipeline (180) and an audio digital signal processor (156). Cartoon lighting and other non-photorealistic effects can be produced by using a lighting calculation to produce a parameter other than color or opacity for use in a later modification of a color or opacity value. In more detail, the example embodiment uses the lighting calculation to generate texture coordinates used in a texture mapping operation (700a). The texture mapping operation (700a) allows lighting computation results to select between brush strokes for cartoon lighting and other effects. The resulting dynamic cartoon lighting animation can be performed on a low cost platform such as a home video game system or personal computer.

EP 1 189 173 A3

Printed by Jouve, 75001 PARIS (FR)

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 01 30 7190

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
X	A. LAKE, C. MARSHALL ET AL: "Stylized Rendering Techniques for Scalable Real-Time 3D Animation" FIRST INTERNATIONAL SYMPOSIUM ON NON-PHOTOREALISTIC ANIMATION AND RENDERING, JUNE 05-07, 2000, - 7 June 2000 (2000-06-07) pages 13-22, XP002263468 Annecy, France sections 3.1, 4.3 * abstract * ---	1,2,4-21	G06T15/50 G06T11/00 G06T15/20 G06T15/00
Y		3	
X	MCCOOL M D ET AL: "TEXTURE SHADERS" PROCEEDINGS 1999 EUROGRAPHICS / SIGGRAPH WORKSHOP ON GRAPHICS HARDWARE. LOS ANGELES, CA, AUG. 8 - 9, 1999, SIGGRAPH / EUROGRAPHICS WORKSHOP ON GRAPHICS HARDWARE, NEW YORK, NY: ACM, US, 8 August 1999 (1999-08-08), pages 117-126,144, XP000977118 ISBN: 1-58113-170-4 section 4.2 ---	1,2,4-8	
X	BENNEBROEK K ET AL: "Design principles of hardware-based phong shading and bump-mapping" COMPUTERS & GRAPHICS, MARCH-APRIL 1997, ELSEVIER, UK, vol. 21, no. 2, pages 143-149, XP004064042 ISSN: 0097-8493 sections 2.1, 2.2, 3.1, 3.2; table 1 * abstract; figure 8 * --- -/--	1,4-8,22	

TECHNICAL FIELDS SEARCHED (Int.Cl.7)
G06T

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
MUNICH	1 December 2003	Meinl, W

CATEGORY OF CITED DOCUMENTS

X particularly relevant if taken alone
Y particularly relevant if combined with another document of the same category
A technological background
O non-written disclosure
P intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 01 30 7190

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
Y	SEGAL M ET AL: "Fast shadows and lighting effects using texture mapping" SIGGRAPH '92. 19TH ANNUAL ACM CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, CHICAGO, IL, USA, 26-31 JULY 1992, vol. 26, no. 2, pages 249-252, XP000600021 Computer Graphics, July 1992, USA ISSN: 0097-8930	3	
A	section 3.2		

A	KLEIN A W ET AL: "Non-photorealistic virtual environments" COMPUTER GRAPHICS PROCEEDINGS. ANNUAL CONFERENCE SERIES 2000. SIGGRAPH 2000. CONFERENCE PROCEEDINGS, PROCEEDINGS OF SIGGRAPH'00: 27TH INTERNATIONAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES CONFERENCE, NEW ORLEANS, LA, USA, 23-28 JUL, pages 527-534, XP001003594 2000, New York, NY, USA, ACM, USA ISBN: 1-58113-208-5 * abstract *	13	

A	US 5 687 307 A (YAMADA SHIGEKI ET AL) 11 November 1997 (1997-11-11) * abstract; figures 9-11 * * column 12-13 *	1-37	

TECHNICAL FIELDS SEARCHED (Int.Cl.7)

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
MUNICH	1 December 2003	Meinl, W

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 01 30 7190

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

01-12-2003

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5687307 A	11-11-1997	JP 3453410 B2	06-10-2003
		JP 7093587 A	07-04-1995
		JP 7110873 A	25-04-1995
		JP 3441804 B2	02-09-2003
		JP 8083352 A	26-03-1996
		JP 7239750 A	12-09-1995

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82